

SOMBRAS EN INTERNET

Sniffer ETHEREAL:

★ Estableciendo Filtros

★ Capturando Información "Sensible"



ATACANDO LA CACHE DNS



MICROSOFT DICE: WINDOWS no depende DE NETBIOS

MICROSOFT MIENTE !!! y nosotros TE LO DEMOSTRAMOS !!!

INSTALAREMOS NUESTRO PROPIO SERVIDOR DNS

SUPLANTAREMOS EL SERVIDOR DE LA RED POR EL NUESTRO

HACK X CRACK - HACK X CRACK - HACK X CRACK - HACK X CRACK

NUMERO 26



BURLANDO FIREWALLS E IDS

ALGORITMOS: FRAGMENTACIÓN Y REENSAMBLADO

Nº 26 -- P.V.P. 4,5 EUROS

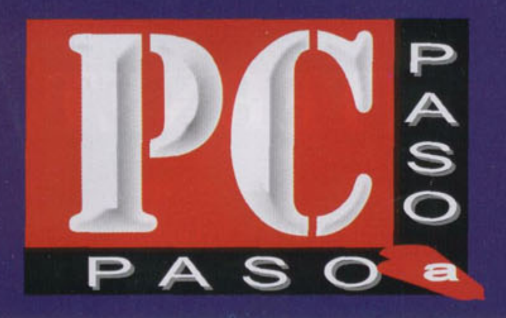


CURSO DE TCP / IP

PROBLEMAS DE SEGURIDAD DERIVADOS DE LA FRAGMENTACIÓN DE DATAGRAMAS

ATAQUES DE FLOOD DE FRAGMENTOS

PC PASO A PASO: EL misterioso ARCHIVO DE WINDOWS services.exe



LOS CUADERNOS DE HACK X CRACK

www.hackxcrack.com

EDITORIAL: EDITOTRANS S.L.
C.I.F: B43675701
PERE MARTELL Nº 20, 2º - 1ª
43001 TARRAGONA (ESPAÑA)

Director Editorial
I. SENTIS
E-mail contacto
director@editotrans.com
Título de la publicación
Los Cuadernos de HACK X CRACK.
Nombre Comercial de la publicacíón
PC PASO A PASO
Web: www.hackxcrack.com
Dirección: PERE MARTELL Nº 20, 2º - 1ª.
43001 TARRAGONA (ESPAÑA)

INDICE

INDICE DE ANUNCIANTES

Director de la Publicación
J. Sentís

E-mail contacto
director@hackxcrack.com

Diseño gráfico:
J. M. Velasco

E-mail contacto:
grafico@hackxcrack.com

Redactores
AZIMUT, ROTEADO, FASTIC, MORDEA, FAUSTO, ENTROPIC, MEIDOR, HASHIMUIRA, BACKBONE, ZORTEMIUS, AK22, DORKAN, KMORK, MAILA, TITINA, SIMPSIM... ... ...

Contacto redactores
redactores@hackxcrack.com

Colaboradores
Mas de 130 personas: de España, de Brasil, de Argentina, de Francia, de Alemania, de Japón y algún Estadounidense.

E-mail contacto
colaboradores@hackxcrack.com

Imprime
I.G. PRINTONE S.A. Tel 91 808 50 15

DISTRIBUCIÓN:
SGEL, Avda. Valdeparra 29 (Pol. Ind.)
28018 ALCOBENDAS (MADRID)
Tel 91 657 69 00 FAX 91 657 69 28
WEB: www.sgel.es

TELÉFONO DE ATENCIÓN AL CLIENTE: 977 22 45 80
Petición de Números atrasados y Suscripciones (Srta. Genoveva)
HORARIO DE ATENCIÓN: DE 9:30 A 13:30 (LUNES A VIERNES)

NUMERO 26 -- PRINTED IN SPAIN
PERIOCIDAD MENSUAL
Deposito legal: B.26805-2002
Código EAN: 8414090202756



SOMBRAS EN INTERNET

- Sniffando La Red - Formas de Detección - Utilidades para la Deteccion
- Ethereal: Descripción. Capturando Tráfico. Estableciendo Filtros
- Capturando Informacion Sensible - Telnet vs Secure Shell

Sniffers, espiando la red.

Puede resultar muy curioso pensar en la cantidad de información que circula por Internet en un momento dado, pero todavía más increíble resulta ver lo poco en serio se toma mucha gente la seguridad informática... cada segundo hay una gran cantidad de tráfico circulando delante de nuestras narices de forma insegura, tráfico que *cualquiera* puede capturar sin excesivo esfuerzo, esto incluye todo tipo de información sensible como contraseñas de cuentas de correo, servidores ftp, web, cuentas de usuario, datos bancarios e información personal que por supuesto no queremos que nadie pueda obtener sin nuestro consentimiento.

Pero ¿qué es exactamente un sniffer? ¿En qué consiste el "Sniffing"? Para decirlo de una manera simple consiste en robar y espiar digo - monitorear - el flujo de información que circula libremente por la red...

Maneras para sniffar una conexión hay muchas y aquí y ahora las vamos a ver Pero te hablaré de otra gran cantidad de cosas y por supuesto de la manera para protegerte de estos "voyeurs" informáticos que lejos de lo que pueda parecer no es tarea fácil.

De hecho, aunque a algunos les cueste creerlo, los sniffers o analizadores de tráfico están muy implantados en Internet, violando la privacidad de las personas en la red. Como claro ejemplo está el caso del DCS1000 (alias "Carnivore") del FBI, una especie de "supersniffer" que al mas puro estilo "gran hermano" nos vigila a todos. Por todo ello, debemos estar preparados para protegernos en la medida de lo posible y cifrar siempre todas nuestras comunicaciones. Jejeje Me parece que tenemos más de un lector especialmente curioso que ya se está frotando las manos... será mejor que empecemos con el artículo

1.- Nociones fundamentales

Nosotros vamos a tratar el tema principalmente en el marco de lo que son las redes Ethernet, por lo que antes de ponernos a "curiosear" debemos tener muy claros algunos conceptos, ya que no es bueno ir por ahí sin saber lo que se hace... ante todo profesionalidad (Muchos de estos temas ya se han tratado en números anteriores de la revista por lo que os aconsejo echarles un vistazo).

La red Ethernet

La red Ethernet fue desarrollada por la compañía Xerox pionera en las redes LAN (redes de área local) que se puede implementar básicamente utilizando cable coaxial, fibra óptica o par trenzado. Existen diversas topologías disponibles, por ejemplo, cuando se utiliza la fibra óptica la topología será de estrella y cuando usemos cable coaxial la topología será de bus.

La idea fundamental: una red Ethernet nos permite de alguna manera poder compartir una conexión y poder conectar varias computadoras entre si.

Protocolo ARP

Al enviar un paquete de información la cabecera de dicho paquete contiene la dirección de la máquina destino. De hecho, para que dos host's en una red local se puedan comunicar es necesario realizar una asociación entre las **direcciones de red (direcciones IP)** y las **direcciones físicas (direcciones MAC).** Serán las direcciones físicas (MAC) de las tarjetas de red las que identifican unívocamente a un host (PC) en una red de área local.

Para asociar direcciones de red (IPs) con direcciones físicas (MACs) existe un protocolo llamado "*Address Resolution Protocol*" o Protocolo de Resolución de Direcciones (desde ahora *ARP*) y que dentro del modelo OSI se sitúa en *el nivel de enlace* o acceso a la red.



Modelo OSI...

Modelo OSI... en el curso TCP/IP y la serie RAW de esta misma revista se ha tratado en profundidad este tema. Si eres un nuevo lector, puedes pedir los números atrasados en www.hackxcrack.com

¿Cómo funciona? Bueno, en la práctica cada host (PC) contiene una tabla en la que se asocia una dirección de red (IP) con una dirección física (MAC) y, mediante ARP se va actualizando dinámicamente usando *broadcast* a la red.

"broadcast" ---> Simplificando mucho y para los que no siguen habitualmente la revista, broadcast es cuando un host (PC) emite una petición a todos los demás host (PCs) de la red.

Si un **host A** quiere comunicarse con otro **host B** buscará en su tabla ARP la dirección IP del host B y a partir de ahí:

- Si tiene la IP en su tabla, puede obtener la dirección física (MAC) asociada y comunicarse.

- Si no la tiene, mal asunto, tendrá que enviar una petición ARP a toda la red (broadcast) indicando en esa petición la IP de la cual solicita la dirección física (MAC).

Cuando un host (**host B**) recibe la solicitud y ve que en esa solicitud figura su IP, actualizará su propia tabla ARP e indicará al solicitante (**host A**) su dirección física (MAC). Ahora el **host A** actualiza su tabla ARP y ya puede comunicarse con el **host B** sin problemas.

Es importante saber que el resto de host's (PCs) de la red que recibieron la petición ARP ignoran el paquete, solo responderá el que tenía la IP consultada por el host A.

Tened en cuenta que los mensajes ARP enviados así, lo hacen en el campo de datos de las tramas Ethernet. Aunque yo siempre os hablo aquí de paquetes (a lo bestia) con el fin de simplificar las explicaciones. Por último comentar que existe una variante del ARP llamado *"Reverse ARP"* que lo que hacer es permitir a una máquina obtener su IP conociendo simplemente su dirección física. Os dejo aquí el formato del mensaje ARP.

DIRECCIÓN MAC ORIGEN	
DIRECCIÓN MAC ORIGEN	DIRECCIÓN IP ORIGEN
DIRECCIÓN IP ORIGEN	DIRECCIÓN MAC DESTINO
DIRECCIÓN MAC DESTINO	
DIRECCIÓN IP DESTINO	

Formato del mensaje ARP

MAC y direcciones físicas

Hemos hablado de direcciones físicas (MACs). Vamos a detallar el tema un poco más.

"*Media Access Control*" (MAC) o Control de Acceso al Medio es una especie de sello que tienen TODAS las tarjetas de red. Ese "sello" son unos numeritos que identifican a cada tarjeta de forma unívoca, es decir, no existen en el planeta dos tarjetas de red con la misma MAC.

Para comprender esto bien nada mejor que destripar una dirección MAC. Bajo entornos Linux nos basta con ejecutar *"ifconfig"* como usuario privilegiado para obtener la información sobre las interfaces de red y configurarlas.

Si trabajamos en un entorno Window$ XP, 2K o 2k3 abriremos una Ventana de Comandos (Menú Inicio ---> Todos los Programas ---> Accesorios ---> Símbolo del sistema **o** Menú Inicio ---> Ejecutar ---> escribir cmd.exe y aceptar) y teclearemos la siguiente orden:

```
C:\>ipconfig /all
```

A continuación se te mostrará mucha información y si dispones de un adaptador Ethernet podrás observar, entre otras cosas, algo *similar* a esto:

```
Dirección física. . . . . . . : 00-0C-29-17-D3-C4
```

En este caso se trata de la dirección MAC de mi tarjeta de red. Como ves las direcciones MAC se expresan como 12 dígitos en formato hexadecimal y tienen una longitud de 48 bits de los cuales los 6 primeros comenzando por la izquierda, esto es, 00-0C-29 (24 bits) identifican al fabricante y son un identificador único organizativo.

Aunque raramente, a veces a las direcciones MAC se les llama BIA ("Burned-in") ya que están grabadas en la memoria ROM de la tarjeta NIC.



¿Puede "falsearse"...

¿Puede "falsearse" (modificarse) la MAC de una Tarjeta de Red? La respuesta es SI, y es lo más fácil del mundo.

Normalmente las tarjetas ethernet que compras vienen con unos drivers. Si instalas esos drivers, generalmente añadirán opciones a la configuración de la Tarjeta, y una de ellas es precisamente "falsear" la MAC.

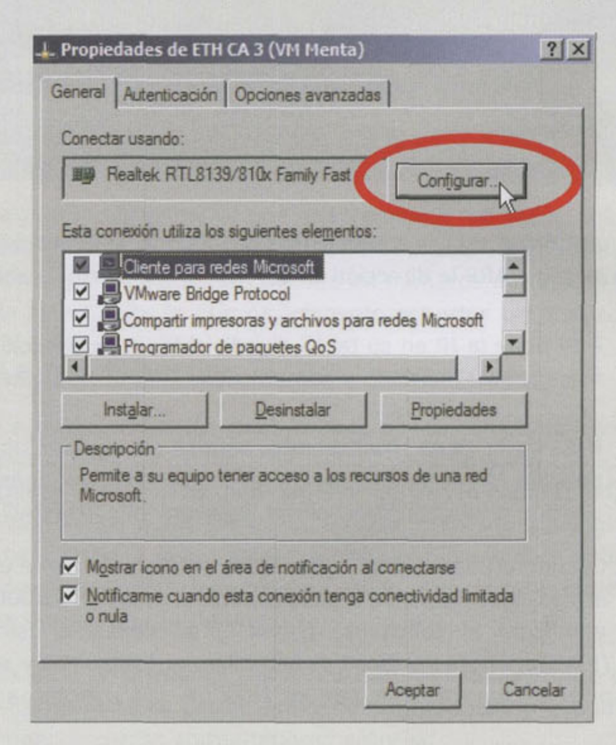
Vamos a verlo... Abre las Conexiones de Red de tu Windows (en un Windows XP --->> Menú Inicio ---> Panel de Control ---> Conexiones de Red e Internet ---> Conexiones de Red).

Verás algo como esto (a mí me salen muchas, pero tú quizás solo tengas una o dos ☺*)*



Esas son todas las tarjetas de red que tengo instaladas en el PC desde el que escribo este artículo. Pulsa sobre cualquiera de ellas con el botón derecho del Mouse y en el menú contextual que aparecerá selecciona "propiedades"

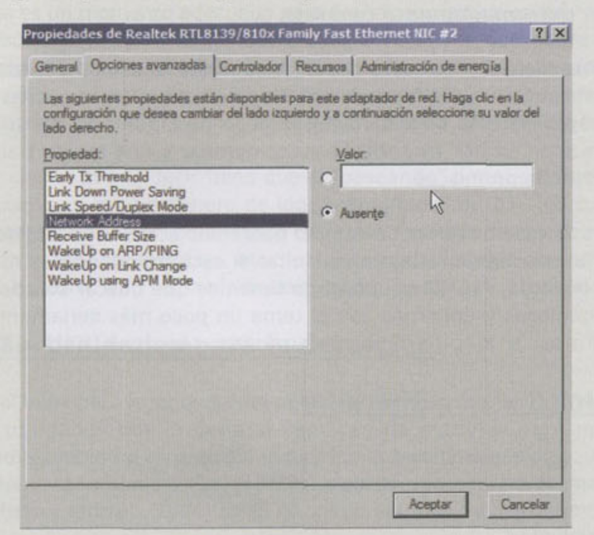
Se abrirá una ventana parecida a esta. Pulsa CONFIGURAR (mira el Mouse en la imagen).



Se abrirá una ventana con varias pestañas... selecciona la pestaña "Opciones Avanzadas" y verás algo parecido a esto:

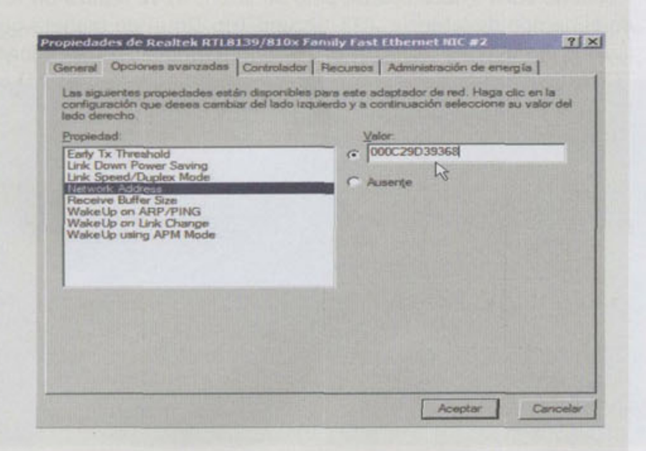
Aquí puedes ver la configuración de la tarjeta de red. Cada tarjeta (según el fabricante y los drivers) ofrecerá opciones distintas, pero TODOS (si la tarjeta es

mínimamente potente) tendrán una opción llamada algo así como "Network Address". Pulsa sobre esa opción y verás la siguiente pantalla:



En la imagen puedes ver que está seleccionada la opción "Ausente" y no hay nada en el recuadro "Valor". Esto significa que la Tarjeta de Red tiene la MAC que por defecto le ha puesto el fabricante y puedes saber cual es tal como te hemos indicado (ipconfig /all). Las MAC, no nos cansaremos de repetirlo, las pone el fabricante y SON ÚNICAS para cada tarjeta de red fabricada en el planeta. No existen dos tarjetas de red con la misma MAC.

Si tú seleccionas la opción "Valor" e introduces otra MAC, a partir de ese momento esa tarjeta de red tendrá la MAC que tú le has puesto... habrás falseado la MAC. La siguiente imagen muestra la introducción de una MAC para nuestra tarjeta.



Cuando has hecho un "ipconfig /all" habrás visto que la MAC figura como pares de valores separados por un guión. Si quieres falsear una MAC en Windows, deberás introducir los valores y ponerlos en el campo "valor" pero sin guiones, todo seguido (fíjate en la imagen).

Un par de apuntes. Si no tienes ni idea de qué números son válidos para una MAC, puedes conseguir los que quieras instalando el programa VMWARE y creando tarjetas de red virtuales. Este programa sirve para crear máquinas virtuales y será tratado en profundidad el mes que viene.

Si te estás preguntando qué utilidad tiene esto de falsear la MAC, puedo darte una muy interesante (y muy legal, es para lo que yo utilizo esta característica

Yo tengo contratada para acceder a Internet una línea de cable con AUNA. En mi contrato entraba un IP Pública Fija y esa IP me la asignaron a una MAC de una tarjeta de red que tenía en mi PC. Un buen día decidí sustituir la tarjeta por otra y llegó el problema

Compré la nueva tarjeta y la instalé, pero la nueva tarjeta tenía una MAC distinta (como debe ser). Al conectarme a la línea de cable recibí una IP distinta... el servicio técnico de AUNA decía que tardaría una semana en reasignarme la antigua IP a la nueva MAC... pues nada, les dije que no se molestasen, que ya arreglaría yo el problema.

¿Cómo arreglé el problema? Pues ya lo sabes, falseando la MAC de la nueva tarjeta tal y como hemos visto. Puse la MAC de la antigua tarjeta en el campo Network Address de la nueva.

Como tema interesante, debo decir que AUNA me da hasta tres IPs públicas por conexión. Una IP pública FIJA (nunca cambia) y dos IPs Públicas dinámicas (que cambian constantemente)... actualmente. Pues gracias al VMWARE "jugueteo" con todas ellas

Ahora bien, el siguiente punto es realmente importante, las tarjetas de red tienen dos formas fundamentales de trabajar:

MODO NORMAL: esta es la forma aburrida que tiene de trabajar una tarjeta de red (desde nuestro punto de vista jejeje) de manera que sólo va a procesar los paquetes que tengan como destino a ella misma.

Es decir, un paquete que se mande desde A hacia C pasando por un host B únicamente será procesado por el host C, el host B hará caso omiso del paquete recibido ignorándolo. Es el modo "natural" de trabajar de una tarjeta de red.

MODO PROMISCUO: aunque bastante explícito se trata de un modo en el que la tarjeta escucha todos ¡atención! *TODOS* los paquetes que pasen por ella... ni que decir tiene que esta es la forma de trabajo preferido de una tarjeta para usuarios malintencionados y administradores preocupados

¿Cómo se selecciona el modo de trabajo de la tarjeta? Lo puedes hacer por software

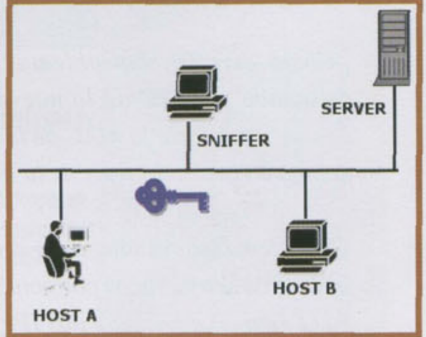
2.- Sniffando la red

¿Qué motivos pueden llevar a una persona a sniffar conexiones en una red? Entre otras:

- Descubrir posibles errores en una red o programa.
- Descubrir ataques en una red.
- Para intentar intrusiones en redes.
- Capturar información sensible, especialmente sino está cifrada.
- Mejorar el rendimiento de la red buscando posibles cuellos de botella.
- Una forma de solucionar una tarde aburrida.

Vamos a ver un esquema de la típica situación en la que tenemos una interfaz en modo promiscuo capturando todo lo que pasa por ella. En el ejemplo tenemos un segmento de red con tres ordenadores conectados a la red y una de las máquinas capturando el tráfico. Cualquier contraseña que viaje no cifrada será visible para el atacante así como otros datos.

Se trata pues de un ataque *pasivo* y por ello se hace a veces difícil la detección de estos comportamientos irregulares en una red e Internet. El atacante tiene dos posibilidades de trabajo aquí, puede acceder y recuperar los logs (información capturada) almacenados por el sniffer desde el exterior o bien procesar la información en tiempo real.



2.1.- Algunas formas de detección

Desde hace bastante tiempo hay unos mecanismos básicos para saber si algo no va como se supone que debería funcionar. Podemos detectar la presencia de un sniffer de muchas maneras y usando una gran variedad de programas; pero también mediante las siguientes técnicas fundamentales que ahora veremos... no obstante, la detección de un sniffer camuflado a conciencia puede ser ***extremadamente*** compleja.

En el mejor de los casos podemos consultar el estado y modo de trabajo de las interfaces de red sin más herramientas que las del propio sistema operativo. En el caso de Linux nos basta con ejecutar **' *root# ifconfig* '** para comprobar si una tarjeta trabaja en modo promiscuo.

En un caso no tan optimista NO podremos acceder directamente a la interfaz de red para consultar si está trabajando en modo promiscuo, etc. Es aquí cuando tenemos que buscar soluciones ingeniosas y ponernos con el tema un poco más seriamente... vamos a explicar unas cuántas técnicas conocidas:

Ping Test o Etherping

Es posible que en ciertos sistemas, debido a un problema interno, podamos elaborar un paquete "ICMP Echo" con la IP de la máquina a verificar pero con una dirección física (MAC) distinta y errónea a voluntad.

La máquina debería descartar el paquete puesto que se trata de una MAC inexistente pero al estar la interfaz trabajando en modo promiscuo dice " hey! Esto también es mío trae 'pa aquí !" y responde a la petición lo que hace saltar la liebre... es lo que tienen algunos sniffers, que son muy posesivos (otros no lo son tanto)

DNS Test

Consiste básicamente en generar una serie de conexiones TCP contra host inexistentes, por lo que cuando el sniffer intente hacer una resolución DNS parea saber el nombre del host, dicho proceso podrá ser detectado por una herramienta de detección de sniffers para discriminar máquinas activas funcionando con interfaces en modo promiscuo.

Latency Test

Mediante ICMP (hacemos un ping de latencia) se realiza un test en el tiempo de latencia RTT (Round Trip Time) de manera que cuando el sniffer esté procesando una gran cantidad de conexiones TCP (que habremos generado especialmente para la ocasión) en un periodo de tiempo muy corto se incremente el tiempo de latencia. Tras una serie de mediciones podremos llegar a determinar si hay un sniffer funcionando en una máquina concreta.

ARP Test

Antes, cuando te he explicado el ARP, es posible que te hayas dado cuenta de que si una máquina que NO esta en modo promiscuo recibe una petición ARP que no es para ella, nunca la verá pero... ¿qué pasa si la interfaz trabaja en modo promiscuo? En ese caso sí que procesaría la petición ya que la interfaz recibiría la solicitud y al considerarla como legítima hará que la máquina responda.

El test consiste simplemente en enviar a nuestro objetivo una petición ARP con una MAC errónea de manera que si responde tenemos una interfaz trabajando en modo promiscuo y una máquina con un posible sniffer activo

Este es un momento adecuado para mencionar la técnica del "ARP poisoning" (que ya se ha tratado en la revista y que ahora no explicaré), altamente efectiva en redes conmutadas. A continuación veremos una serie de utilidades básicas y no tan básicas para la detección de sniffers e incluso envenenamiento ARP en una red.

Por cierto, ni que decir tiene que si "paseando" por el sistema te encuentras con un fichero de logs "camuflado" de 237 MB cuyo contenido es claramente tráfico de la red, cuentas de correos en texto plano, contraseñas, etc... entonces significa que un sniffer ha estado activo ahí desde hace bastante tiempo

2.2.- Utilidades para la detección

Efectivamente, llegados a este punto os voy recomendar una serie de utilidades por lo general sencillas de manejar pero muy funcionales - tanto para Window$ como para Linux - para que todos quedemos contentos... *aquí no queremos discriminar a* nadie

Por desgracia no te puedo detallar de forma exhaustiva cada una de estas herramientas *porque no es el objetivo (la mayoría de* ellas tienen un manejo sencillo con interfaces bastante intuitivas). Lo que si que haré en breve es explicarte el funcionamiento de una gran herramienta que es importante conocer... pero primero y "de pasada" te presento las *siguientes*:

"Ettercap"

Empezamos con una potente aplicación que tenéis disponible en varias plataformas incluyendo Window$ XP/2K/23K, FreeBSD, OpenBSD, NetBSD, Linux 2.4.x / 2.6.x y Solaris ¿genial no?

Desde Linux podemos verlo con el "look&feel" que más nos guste ya que debemos indicarle con que tipo de interfaz queremos lanzar el programa:

root# ettercap -C para el modo basado en ***ncurses***
root# ettercap -G para el modo basado en ***GTK***

Como decía, Ettercap permite hacer muchas otras cosas aparte de capturar, inspeccionar e inyectar tráfico a la red.

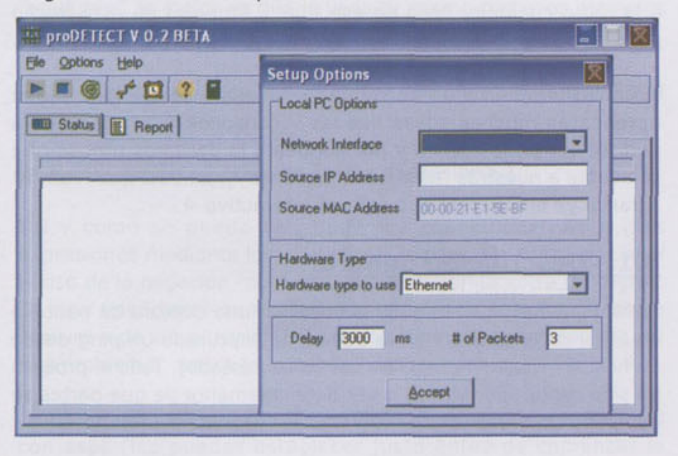


¿De dónde bajarlo? Prueba http://sourceforge.net/projects/ettercap/

"Prodetect"

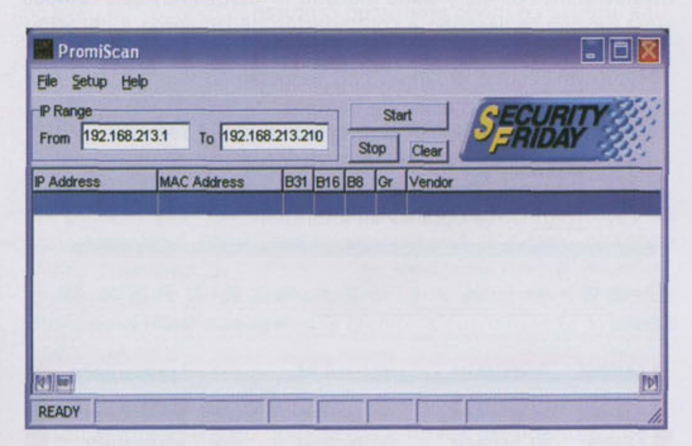
Una sencilla aplicación para Window$ que te será muy útil es "Prodetect" y la tienes disponible de forma gratuita aquí: http://sourceforge.net/projects/prodetect/

Tras descargarlo y ejecutarlo se os mostrará el "InstallShield" típico para la instalación en la que os podéis limitar a pulsar en "Siguiente" cuando os aparezca.



"Promiscan"

Se trata de una sencilla aplicación que funciona bajo Window$ que podéis obtener también de forma gratuita desde la url de interés *http://www.securityfriday.com*. Requiere WinPcap instalado en el sistema para poder funcionar, *lo tienes disponible en http://winpcap.polito.it*



3.- Ethereal

Ethereal es un potentísimo analizador de protocolos que tenemos a nuestro alcance de forma gratuita tanto para entornos Linux como Window$. Lo tienes disponible para bajar plenamente funcional desde la siguiente URL: http://www.ethereal.com/download.html

En el caso de Window$ necesitarás tener instalado WinPcap en tu ordenador, una vez más lo tienes disponible aquí http://winpcap.polito.it. La instalación de Ethereal y WinPcap es realmente sencilla por lo que no lo voy a explicar aquí, te bastará con seguir las instrucciones que el instalador te indicará en el caso de Window$. En el caso de Linux y dependiendo del sabor que uses se hace de manera habitual a como instalarías cualquier otro tipo de software mediante paquetes o código fuente.

Bien, ahora ya sabes de donde obtenerlo vamos a saber cómo usarlo para múltiples propósitos. En ningún caso hay que pensar que Ethereal puede manipular el tráfico para enviar luego paquetes a la red o cualquier cosa de ese tipo... Ethereal no está hecho para eso sino para hacer un análisis exhaustivo del tráfico.

Nosotros veremos esta herramienta - paso a paso - con la que aprenderás muchas cosas nuevas y consolidarás otras que ya pudiste ver en la revista y por supuesto le daremos un uso que se adapte a nuestras "oscuras intenciones" jeje... así que ¡vamos! Arranca ya el Ethereal que esto es interactivo

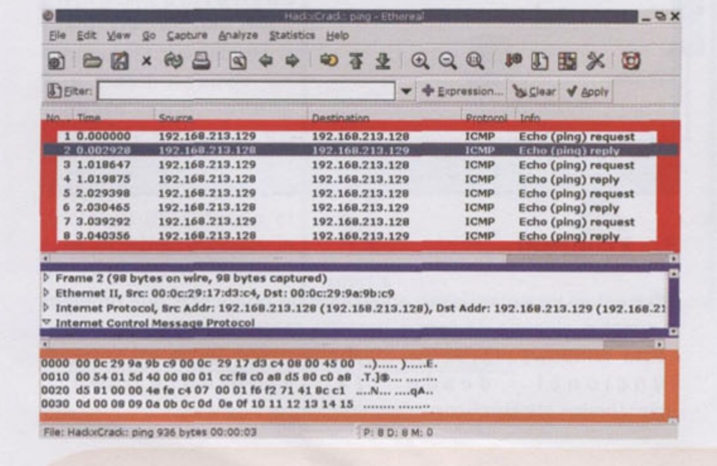
3.1.- Descripción general

En la siguiente imagen puedes observar una captura de pantalla de Ethereal bajo Linux en la que se ha capturado un ping desde un host A hacia un host B (el cual ha contestado). Todo el proceso ha sido capturado. Vamos a ver detalladamente de que partes se compone Ethereal:

En primer lugar tenemos toda la parte de los menús con las diversas opciones y los iconos que permiten acceder de forma rápida a algunas de esas opciones que luego vamos a ir viendo.

Por otro lado tenemos la zona que te he marcado de rojo en la que se muestra la captura en sí con la información de interés inmediato como lo son las distintas IP's, los puertos y protocolos involucrados, etc. Tal y como muestra la imagen hay seis campos cuya función te describo a continuación de izquierda a derecha:

(1) No.: indica el número de paquete.
(2) Time: información de tiempos.
(3) Source: Dirección de origen
(4) Destination: Dirección de destino
(5) Protocol: Protocolo
(6) Info: Otros detalles de interés



En el caso de la captura habrás observado que efectivamente el host 192.168.213.129 envía un ICMP Echo Request hacia 192.168.213.128 que a su vez responde con un ICMP Echo Reply, ya que hemos hecho un ping a ese host.

Finalmente comentar que si te vas a *"Edit -> Preferences (*desde *'columns')"* puedes mostrar u ocultar la información que se muestra en esos seis puntos. Puedes acceder a este menú de opciones a través de la combinación "Shift + Ctrl + p" o con el penúltimo icono que se muestra en la imagen.
Antes de pasar a la siguiente zona me gustaría que fueses a la opción de estadísticas en el menú justo en la parte de arriba donde dice *"Statistics -> Summary"* y *"Protocol Hierarchy"* observarás una serie de estadísticas organizadas y ampliamente detalladas sobre la sesión o captura actual

Ahora si te fijas en la parte marcada de azul puedes ver que se nos muestra cada vez la estructura del mensaje seleccionado arriba de manera que puedes ver de forma organizada cada una de las cabeceras de los protocolos empleados por el mensaje.

Puedes ir haciendo 'clic' en las flechas para ir tapando o destapando información adicional sobre esa cabecera. De esta forma puedes observar en el ejemplo cómo aparece ese "Internet Control Message Protocol" (ICMP) enviado y desplegarlo en su opción para mostrar información sobre su contenido, como puede ser el tipo (echo reply / echo request ...) , número de secuencia, checksum, etc.

Finalmente, en la parte de abajo marcada de naranja puedes ver el paquete volcado en formato hexadecimal, por si aún te queda alguna duda con la información que te dan arriba Esta parte la ocultaremos, al menos de momento, para trabajar con mayor espacio y cómodamente. Desde "*View -> Packet Bytes*" puedes ocultar y mostrar el volcado hexadecimal, lo mismo podemos hacer con el resto de partes.

Acerca del menú "*File*" es importante saber que puedes guardar y abrir capturas previas en diversos formatos (el indicado por defecto "libcap" es el recomendado). Si deseas guardar la sesión actual ves a "*File -> Save as...*" y si deseas abrir una captura existente usa "*File -> Open*". Esto no tiene misterio, pero lo que sí que puede interesar es la posibilidad de exportar la información a otros formatos como texto plano, Postscript o XML desde "*File ->Export ...*".

3.2.- Capturando tráfico...

Bueno... ahora empieza la parte que nos interesa, y es saber cómo narices empezamos a capturar todo lo que circula por la red ¿verdad?

De nuevo te pongo una imagen para explicarte con detalle las posibilidades que Ethereal nos ofrece. Para empezar una nueva captura pulsa "Ctrl. + K" o bien desde al menú "*Capture -> Start*" con esto te aparecerá una pantalla dividida en cinco "frames" que te describo en seguida:

En el apartado "Capture" tienes la posibilidad de escoger una interfaz existente y válida desde la que realizar la captura, puedes

indicar el tamaño de los paquetes a procesar y establecer filtros (que veremos luego) a la captura. Por supuesto tienes un "checkbox" para capturar en modo promiscuo.

El apartado "Capture file(s)" es usado básicamente para dirigir la captura a fichero o varios indicados aquí, nosotros realizaremos las capturas en tiempo real por lo que ahora mismo no es de nuestro interés.

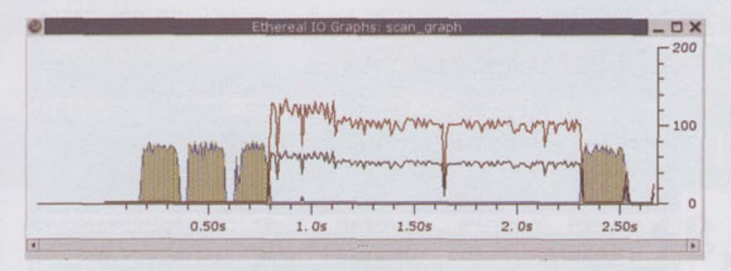
Desde "Display Options" podemos hacer que la captura en tiempo real se vaya mostrando y actualizando y otras cosas:

- ocultar la ventana de diálogo de la captura donde se va viendo lo que vamos capturando según protocolo
- el número de paquetes capturados
- Y el porcentaje con respecto al total (por lo general lo dejaremos siempre con sus valores por defecto).

Podemos decirle a Ethereal que pare de capturar tráfico desde "Stop capture" e indicando una serie de límites que pueden ser (en estricto orden de aparición) por número de paquetes, MBytes totales procesados o cierto periodo de tiempo expresado en horas, minutos o segundos.
Por último en la sección "Name Resolution" tienes la posibilidad de habilitar o deshabilitar la resolución de nombres para direcciones MAC, red y capa de transporte.
Muy bien, ahora solo falta darle al botón "OK" para empezar a capturar :) Debes parar tú mismo la captura (Ctrl + E o desde el icono apropiado) cuando lo creas conveniente salvo que hayas programado Ethereal para hacerlo automáticamente (como te he comentado hace un momento).

Si bien en nuestra muestra de tráfico inicial la lectura es bastante inteligible ya que simplemente había ocho paquetes icmp (4 icmp echo request y sus respuestas echo reply), a veces, debido a la gran cantidad de tráfico capturado y la diversidad de protocolos involucrados en los mensajes la lectura de dichos mensajes puede ser complicada.

De hecho, una gran cantidad del tráfico capturado - sin aplicar ningún tipo de elemento discriminante - no será en muchos casos de interés para nosotros. Podemos ilustrar esto en el siguiente gráfico generado por el propio Ethereal ("*Statistics -> IO Graphs*"), las coordenadas 'x' e 'y' representan tiempo y paquetes respectivamente.



Desde una máquina se han realizado varios escaneos de forma simultánea mediante Nmap (que ya se explicó detalladamente en números anteriores de la revista). En el gráfico puedes ver el escaneo UDP realizado de color rojo. De color negro tienes representados paquetes de tipo ICMP.

La línea y el relleno de color verde se corresponde con paquetes que tienen el "flag SYN" levantado propio de los ataques "SYN Stealth", la línea de azul son paquetes con el "flag RST" levantado lo que tiene lógica debido al escaneo de puertos lanzado contra el host.

3.3.- Estableciendo filtros

Como acabas de ver con Ethereal es posible establecer filtros para discriminar el tráfico o para su visualización con una sintaxis que resultará muy familiar a quienes hayan utilizado alguna vez "tcpdump", lo mejor es que lo veas tu mismo con unos ejemplos sencillos.

La sintaxis de las primitivas, de forma generalizada, la escribimos así:

[not]**<expresión> [and | or** [not] **<expresión>]**

Tal y como se puede ver, podemos concatenar una o más expresiones mediante los operadores lógicos. De forma *opcional* el uso de la negación "not", de ahí que la ponga entre corchetes. Es muy importante que sepas que la sintaxis para los filtros de captura y de visualización NO son los mismos, pero tranquilo, no hay problema, ahora te lo explico paso a paso.

Como he empezado a tratar los filtros de captura empezaremos con esos (los puedes establecer justo antes de comenzar la captura) existen muchas alternativas por lo que veremos solamente algunas básicas:

[origen | destino] <host> :
Permite filtrar el tráfico generado o entrante por un host.

Ethernet [origen | destino] <host> :
Como en el caso anterior pero para direcciones Ethernet.

['< =' | '>='] <longitud> :
Útil para capturar paquetes de tamaño mayor o menor al indicado.

[ip | ethernet] <protocolo> :
Seleccionará paquetes del protocolo indicado en IP o Ethernet.

Se pueden usar los operadores de negación *(`!' o `not')* de concatenación *(`&&' o `and')* y alternación *(`||' o bien `or')* entre varias primitivas.
Buff! Vale, vale ya se que así parece complicado pero no lo es en absoluto... mira te pongo unos ejemplos que quizás te ayuden a verlo más claro:

'tcp port 80 host 127.0.0.1':
Esto captura el tráfico que circule a través del puerto 80 de tu PC.

'src host localhost':
Esto captura el tráfico generado por nosotros.

'tcp port 80 not host 62.193.200.34'
Esto captura el tráfico HTTP en el que no interviene 62.193.200.34

Hasta aquí los filtros de captura, ahora pasamos a los de visualización, pero antes debo comentarte que existen expresiones mucho más complicadas que las de arriba del tipo 'icmp[0] = 8' (selecciona solo el tráfico "ICMP echo request"). Pero por no podemos entretenernos ya que el artículo persigue otros objetivos, en cualquier caso, si estás muy interesado y tu curiosidad no puede esperar te sugiero leer la documentación de "tcpdump".

Una vez más insisto en que esto es interactivo, así que prepara tu navegador de Internet favorito... ¿el IExplorer es tu favorito?... no importa, por esta vez nos vale

Apunta hacia http://www.hackxcrack.com e inmediatamente después inicia una captura con Ethereal, tras navegar durante unos segundos capturando el tráfico...

Observarás una gran cantidad de información. Si seleccionas uno de los paquetes haciendo 'clic' derecho con el ratón te aparecerá un menú contextual con diversas opciones muy interesantes que vamos a ver ahora mismo. Por ejemplo:

- puedes separar en una ventana individual ese paquete para examinarlo más detenidamente si eliges "*Show packet in new window*" desde ese menú contextual.
- también puedes marcar el paquete para resaltarlo con "Mark Packet" y desmarcarlo de igual forma.
- también puedes aplicarlo como filtro desde "*Apply as filter -> Selected* ". Escoge un paquete de tipo DNS para la resolución del nombre hackxcrack.com y observarás como en la parte de arriba parece en "*Filter:*" algo como lo siguiente:

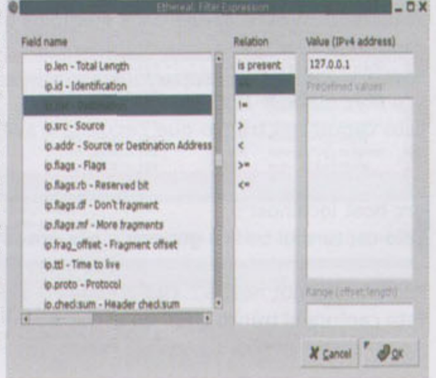
'ip.dst == 195.235.133.3'

Acabas de establecer un filtro usando ese paquete concreto como modelo :) así el resto de paquetes han desaparecido y solamente quedan aquellos cuya IP de destino sea 195.235.133.3. Ahora vete arriba y modifica tu mismo el valor de filtro:

'ip.dst == 62.193.200.34 and http'

Esta expresión hace que solo aparezca el tráfico cuyo protocolo sea HTTP hacia la IP de destino indicada. Podemos también separar aquellos paquetes que no tengan como destino el puerto 8080 de esta manera: '<expresión> and tcp.dstport == 8080' en cualquier momento y desde el icono "*Clear*" puedes quitar los filtros aplicados actualmente a la captura.

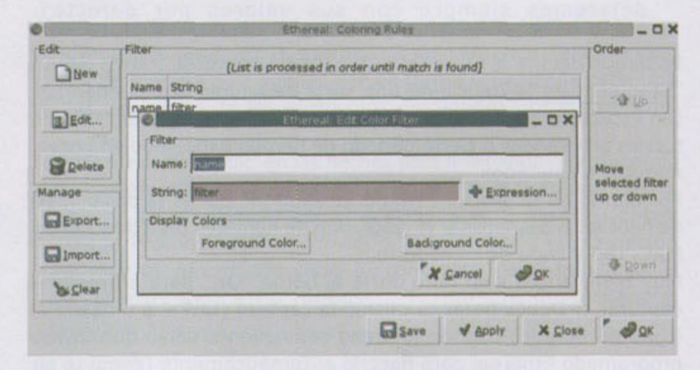
Aunque la sintaxis es muy intuitiva es buena idea usar el asistente que trae Ethereal, lo puedes activar desde la opción "*Expression*", su icono en la barra, o bien, desde el menú "*Analyze -> Display filters -> Expression*" te aparecerá una ventana como la que se te muestra en la imagen.



De entre todos los campos que hay y que son muchos, hay algunos que nos interesan de forma especial como TCP, IP, ICMP, TELNET, SSH, MySQL, HTTP, ARP y otros tantos tu mismo puedes ver que hay para todas las necesidades

Respecto a los operadores tenemos los típicos: mayor, menor, menor o igual, mayor o igual, diferente, igual y además "is present" (está presente), "contains" (contiene) y "matches" ("igual a..." o "encaja con...") con lo que podemos construir nuestras expresiones con total libertad.

El ejemplo que se muestra en la captura es algo como 'ip.dst != 127.0.0.1' que significa "muestra todo lo que NO nos tenga como destino". No es complicado ir ahora añadiendo más reglas ¿verdad?.. podemos complicarlo tanto como queramos.



Finalmente, podemos colorear la salida por pantalla de manera que nos resulte mucho más fácil navegar por nuestro "botín" . Te lo explico rápidamente, ves a "*View -> Coloring rules*" y desde ahí accederás a una pantalla como la de la imagen desde la que puedes editar, crear eliminar reglas e importar / exportar las reglas desarrolladas. Si te fijas las expresiones se crean de igual forma que los filtros visuales con la única diferencia de que podemos asignar un color a la fuente ("*foreground*") y al fondo ("*background*").

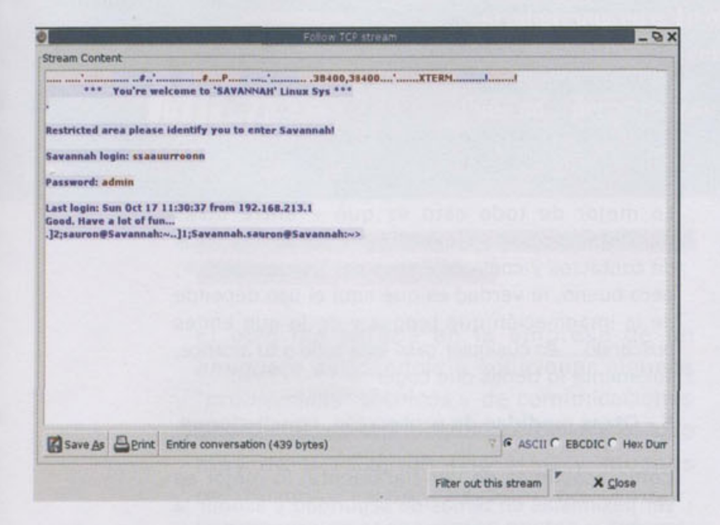
No se a vosotros pero a mi me ha quedado muy bonito, os lo muestro en la imagen que como ves hasta aquí es todo muy general por no decir al azar; pero podemos ser más específicos. Por ejemplo, los paquetes "ICMP Echo" de tipo 8 (reply) y 0 (request) irán con dos tonos de verde distintos, puedes concretar muchísimo pero para ilustrar su funcionamiento nos basta.

No..	Time	Source	Destination	Protocol	Info
86	24.512804	192.168.213.1	192.168.213.129	ICMP	Echo (ping) request
87	24.513208	192.168.213.129	192.168.213.1	ICMP	Echo (ping) reply
90	25.570240	192.168.213.128	192.168.213.254	DHCP	DHCP Request - Transaction ID 0
91	25.590105	192.168.213.254	192.168.213.128	DHCP	DHCP ACK - Transaction ID 0x
92	34.405040	192.168.213.128	192.168.213.2	DNS	Standard query A hackxcrack.co
93	35.416861	192.168.213.128	192.168.213.2	DNS	Standard query A hackxcrack.co
94	36.415739	192.168.213.128	192.168.213.2	DNS	Standard query A hackxcrack.co
95	38.425457	192.168.213.128	192.168.213.2	DNS	Standard query A hackxcrack.co
96	39.594769	192.168.213.2	192.168.213.128	DNS	Standard query response A 62.1
97	39.643803	192.168.213.128	62.193.200.34	TCP	3140 > ftp [SYN] Seq=0 Ack=0 W
98	39.800711	62.193.200.34	192.168.213.128	TCP	ftp > 3140 [SYN, ACK] Seq=0 Ack
99	39.801633	192.168.213.128	62.193.200.34	TCP	3140 > ftp [ACK] Seq=1 Ack=1 W
100	39.951741	62.193.200.34	192.168.213.128	FTP	Response: 220 FTP server (Versi
101	40.052317	62.193.200.34	192.168.213.128	FTP	[TCP Retransmission] Response:
102	40.052739	192.168.213.128	62.193.200.34	TCP	3140 > ftp [ACK] Seq=1 Ack=71 V
103	40.609697	192.168.213.2	192.168.213.128	DNS	Standard query response A 62.1
104	41.413448	192.168.213.128	62.193.200.34	FTP	Request: USER aslkfjklsdfk
105	41.413786	62.193.200.34	192.168.213.128	TCP	ftp > 3140 [ACK] Seq=71 Ack=20
106	41.534495	192.168.213.2	192.168.213.128	DNS	Standard query response A 62.1
107	41.559621	62.193.200.34	192.168.213.128	FTP	Response: 500 Syntax error.
108	41.569356	62.193.200.34	192.168.213.128	TCP	ftp > 3140 [FIN, PSH, ACK] Seq=
109	41.570493	192.168.213.128	62.193.200.34	TCP	3140 > ftp [ACK] Seq=20 Ack=91

File: (Untitled) 42 KB | P: 112 D: 108 M: 2

4.0.- Capturando información sensible

Pues sí, tarde o temprano tenía que pasar ¿no? Con la de tráfico que hay por ahí circulando como no vamos a meter nuestras zarpas jejeje... muchas veces hemos escuchado aquello de que la criptografía es importante para la seguridad y que debemos usar SSH en lugar de TELNET, muy bien, ahora vamos a ver el motivo de una manera bastante contundente.



4.1.- Telnet vs Secure Shell

Muy bien, en el siguiente caso práctico tenemos una máquina corriendo "Telnetd", servicio que corre en el puerto tcp/23 al que nos conectaremos desde otro host cliente mediante Telnet – obviamente –

La gracia reside en el hecho de que hay un tercer invitado (nosotros) que estamos sniffando la red veamos que ocurre:

Restricted area.
Please identify you to enter Savannah!

Savannah login: sauron
Password:
Last login: Tue Oct 19 13:41:45 from 192.168.213.128
Good. Have a lot of fun...
sauron@Savannah:->

OK todo nuestro, aplicamos un filtro a la captura 'telnet and tcp.dstport==23' y nos aparecen un montón de paquetes indicando "*Telnet data...*"

En el campo "*Info*", si haces 'clic' con el botón derecho del Mouse sobre un paquete y eliges la opción "*Follow TCP Stream*" para seguir el flujo TCP en el menú contextual, podrás observar aterrorizado toda la comunicación login y passwd incluidos.

Si ahora hacemos lo mismo con SSH (tcp/22) observamos que no podemos ver *nada* porque la información viaja cifrada una lástima.

4.2.- "Revisando" el correo

Vamos a seguir viendo lo que ocurre cuando enviamos nuestros datos sin más *confiando* en que no hay *nadie* que pueda interceptar esos datos que, al no estar cifrados, son susceptibles a que cualquiera pueda ver su contenido sin dificultad.

Esta vez el proceso que capturamos es una comunicación SMTP que para el ejemplo es la siguiente:

220 Savannah.quasar ESMTP Postfix
helo world
250 Savannah.quasar
mail from: root@savannah
250 Ok
rcpt to: manager@savannah
250 Ok
data
354 End data with <CR><LF>.<CR><LF>
Hola, acabo de dar de alta su cuenta para acceder al código fuente.

Login: mgr
Passwd: sniffer

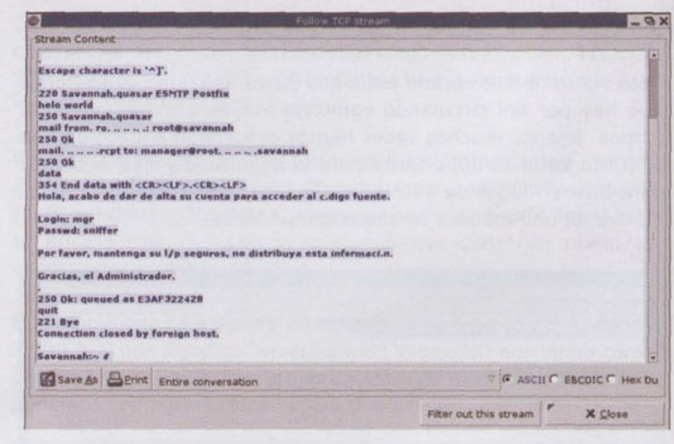
Por favor, mantenga su l/p seguros, no distribuya esta información.

Gracias, el Administrador.

.
250 Ok: queued as 8E1332241B
quit
221 Bye

jejeje no se lo diré a nadie descuida, como en el caso anterior Ethereal ha capturado el tráfico. Tras filtrar y examinar los resultados puedes ver que este mensaje es de todo *menos* privado.

Para ver que eso es cierto prueba a enviar tu mismo un e-mail y una vez con el tráfico en Ethereal haz 'clic' derecho sobre un paquete. Al aparecer el menú emergente selecciona *"Decode as... SMTP"* (decodificar captura como SMTP) y una vez más seleccionaremos aquello de lo que nos interesa seguir el flujo de conversación mediante *"Follow TCP Stream"* y así podemos ver - siempre que no se haya usado cifrado - toda la información del envío como en el ejemplo anterior.



Lo mejor de todo esto es que - entre otras muchísimas cosas - también funciona con las listas de contactos y conversaciones del *"Messenger"*, pero bueno, la verdad es que aquí el uso depende de la imaginación que tengas y de lo que andes buscando... en cualquier caso está todo a tu alcance, solamente lo tienes que coger

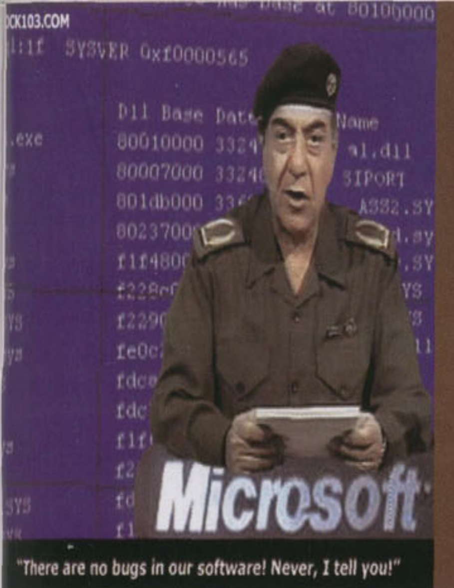
5.- Otras medidas de protección, conclusiones.

Como acabamos de ver claramente, lo mejor es ser pesimistas en temas de seguridad y asumir la realidad de que nuestras comunicaciones diarias son susceptibles a ser *sniffadas* y controladas. Por lo que si queremos privacidad debemos :

- Navegar siempre usando el protocolo seguro http (443/HTTPS) y mediante SSL.
- Usar encriptación del tipo PGP (Pretty Good Privacy) y/o GnuPG para la mensajería instantánea, correo, etc.
- Autentificación y sesiones mediante protocolos seguros y cifrados como SSH o similares.

Mediante el uso de esos protocolos y sistemas de cifrado no podremos evitar que capturen nuestro tráfico pero sí podremos evitar que pueda ser inteligible por un atacante, lo que combinado con sistemas de detección de intrusiones, cortafuegos y la puesta a punto de nuestros sistemas en Internet mitigarán los efectos de una posible intrusión o de un intento de capturar información sensible de tu máquina o red.

No obstante, en la revista hemos visto y seguiremos viendo - paso a paso - como en muchas ocasiones ni siquiera esas medidas son suficientes y es que no hay puerta que no se pueda abrir usando la llave de la imaginación no vayas a perderte los próximos artículos seguro que te sorprenderán.



"There are no bugs in our software! Never, I tell you!"

ATACANDO LA CACHE DNS

Microsoft MIENTE al decir que su "Windows 2000" no depende de NetBIOS

Bufff, me vais a matar... por lo menos aquellos que están interesados en el curso de Redes y Seguridad que venimos publicando desde hace unos meses.... ya.... en el número anterior prometí que en este otro nuevo iba a continuar con el asunto, concretamente con Arquitectura de Redes Wan.... ¿Y no? ¿Hoy tampoco toca?

Pues no.... pero tiene su explicación.... que ahora mismo daré. Ahhh!!! Por si no lo habéis adivinado aún... soy Vic_Thor

Cuéntame lo que haces en un día cualquiera de trabajo....

El otro día estaba en la empresa de un *amiguete* ayudándole a solucionar algunos "*problemillas*" técnicos y de comunicaciones que sufría, es una pequeña empresa con no más de 8 PC's, un trío MAC y muchos periféricos "*raros*", escáner de tambor, impresoras de todas clases gustos y colores, copiadoras digitales conectadas a los PC's, unidades de backup, etc... *Es una imprenta con espíritu y aspiraciones de algo más grande... y lo está consiguiendo,* ***enhorabuena***.

Había muchos problemas, sobre todo en algún MAC y con algunos ordenadores que corrían Windows 95, *sí... todavía existe eso...*

El caso es que uno de los "*chismes*" que tenía como periférico era una especie de lector de tarjetas de crédito que conectaba a uno de sus ordenadores a su *banca online* por Internet...

"Esto es lo más seguro del mundo, cuando me di de alta en el banco me pusieron este cacharro, dispone de cifrado y encriptación, para poder hacer algo contra mis cuentas me tienen que robar el artilugio este.... y aunque lo hagan, todavía tienen que superar el registro en la web por nombre y contraseña, vamos que no hay manera".

Esas fueron sus palabras.... rotundas y definitivas....

Yo le respondí.... *"Claro, claro, tu cría fama y échate a dormir.... mira, no sé si desde Internet podrán o no acceder, pero te aseguro que desde la red local se puede, otra cosa es la forma o el método que se elija..."*

El Sr. de la imprenta, lanza el reto y desafía... *"Pues si tú lo dices, pero a mí me dijeron que las comunicaciones van cifradas, que aunque me pinchen la red local, lo único que verán son caracteres incomprensibles, además... dispongo de un switch que me aconsejó el técnico de la banca online, de ese modo ningún equipo interno podrá escuchar el tráfico de mi equipo con el resto de mi red o por Internet, ¿no es así?"*

Lo que mi amigo dispone es de un "*aparato*" como él dice, que solicita los certificados a su entidad emisora de forma automática, sus comunicaciones van cifradas, efectivamente y tras superar esos "*obstáculos*", por fin llegamos a la bonita web del banco, la cual, cuando deseamos realizar alguna operación, nos pide un nombre *login y password* para acceder al área de usuarios registrados y autorizados.... pero también podemos llegar a esa web "*sin aparato*" aunque tengamos que instalarnos el certificado "*manualmente*"...

Bueno, no os podré toda la conversación, basta con esto para hacernos una idea de lo que pretende este artículo... averiguar su nombre de usuario y contraseña para el acceso a su *banca online*.

El caso es que en esa charla, salieron cosas como keylogers alojados en su equipo, virus, troyanos, puertas traseras, etc... Pero **lo que más le llamó la atención** a este hombre, es que él podría "*navegar*" a la web de su banco y sin saberlo, comerlo, ni beberlo, dirigirse a otra web "*parecida*" o idéntica... y que cuando escriba su nombre y contraseña, "*alguien*" lo reciba y pueda conectarse como si fuese el auténtico usuario... con certificados, sin certificados con aparato o sin él...

NO me estoy refiriendo a mandarle un mail con un link falso, nada de eso... me estoy refiriendo a que SIN TOCAR su equipo y sin necesidad de hacerle caer en ninguna trampa o intrusión en el PC, seamos capaces de redirigirle a otro sitio diferente al que quería acceder de verdad.

Y **de esto tratará este artículo...** para ello **aprenderemos cómo funcionan los Clientes DNS** en general y especialmente los de *Windows* (y digo de *Windows* porque éstos disponen de una funcionalidad "*peculiar*"), de **cómo envenenar la caché de un cliente o servidor DNS** que opera localmente en una *LAN*, de **cómo funciona el protocolo DNS y de muchas otras cosas....**

No pretendo repetir artículos pasados, me estoy refiriendo al artículo publicado en la revista 14 de esta publicación... pero sí que lo vamos a necesitar, y algún otro, gracias también a ti ***moebius***... que por cierto, *¿Dónde te metes que hace tiempo que no te veo?*

PyC (LCo), mediante la serie ***Raw***, nos explicó de forma clara, amena y muy acertadamente cómo funciona el ***protocolo DNS***, algunos ataques contra los mismos y técnicas para ello.

Para el artículo que nos ocupa, ***tomaré prestados*** *conceptos que nos puedan ser útiles, apenas unas pocas líneas, algunos comentarios, quizás no literalmente pero sí con el mismo fondo... es por ello que desde aquí les agradezco a sus autores tan buena fuente de información y que si algo de lo que escribo se parece "sospechosamente" a lo que ya dijeron ellos mismos, es por ésta razón...*

Así que vamos a ello, empecemos por el título de este artículo: ***"Atacando la Caché DNS"***

Mmm, bajo este título tan rimbombante se esconde una pequeña salvedad, que es la caché de clientes DNS o de servidores DNS Locales y no de Servidores DNS públicos, esto es así por varios motivos:

1º) **Es un tema que nunca se ha tocado**, ni en los foros de hackxcrack, ni en esta revista... y a mi modesta opinión, en ninguna de las publicaciones, libros, revistas y sitios que conozco...

2º) **El asalto a Servidores DNS públicos**, además de ser un **grave delito** y por el que ***nos podemos jugar varios años a la sombra en la Institución penitenciaria que nos adjudique el Estado....*** además de eso, que ya es bastante importante, hoy en día es bastante improbable que ocurra con los nuevos servidores ***BIND***, protocolos ***DNSSec***... los ataques de *cumpleaños*, envenenamiento de caché en el servidor, etc... Es algo muy penado y difícil de conseguir... ***ojo... no imposible y si no que se lo pregunten a los creadores del gusano SQLlamer***.

3º) Muchas otras publicaciones, revistas "*de la competencia*", *Webs*, etc... Últimamente se "*han tirado a la piscina*" con artículos sobre ataques contra DNS, hay de todo, desde los que empiezan con buen ánimo y "la explicación del ataque" termina siendo una serie de recomendaciones y probabilidades sin un mero ejemplo, hasta los que te "*cuentan historietas de Mortadelo, Filemón y el FBI*, queda en agua de borrajas y terminan diciendo que te busques la vida y que tengas cuidado.

También los hay serios y técnicamente bien elaborados, pero "*les falta algo*"

Nosotros queremos ser diferentes, ***llevarlo hasta su último extremo, con todas sus consecuencias y sufriendo los rigores del ataque que expondremos más adelante***, pero vamos a ser serios y claros desde el principio, aunque con alguna

"*modificación*" podemos armar una escabechina por Internet o por una red cualquiera, nuestro experimento estará controlado, será efectivo, no será un quiero y no puedo, pero nos limitaremos a clientes locales y servidores locales.

Para poder intentar ataques contra ***servidores DNS***, primero tendríamos que saber cómo configurarlos, cómo funcionan y como se comunican, eso más o menos ya lo expuso PyC en su artículo mencionado, queda mucho que cortar y hablar de esto... en este artículo, terminaremos ***poniendo en marcha un Servidor DNS BIND 9 bajo LiNUX***, un tanto pobre en diseño y seguridad... pero funcional.... *ya veréis para qué lo vamos a usar*... **el mundo al revés, un servidor DNS que "ataca" a sus clientes en lugar de servirles...**

Hoy vamos a aprender muchas cosas de los Servidores DNS, de sus Clientes, de cómo se comunican entre sí, de cómo se transportan los datos y de las características más "*singulares*" del protocolo DNS*, un gran desconocido y a su vez, uno de los protocolos más importantes y usados en la red.*

DNS. Servidores, servicios y protocolo

DNS son las siglas de ***Domain Name Service, Servicio de Nombres de Dominio***, es un protocolo que sin saberlo o no, usamos todos los días cuando nos conectamos a *Internet* y visitamos diferentes páginas... cuando hacemos eso, escribimos en la barra de dirección de nuestro navegador la dirección del servidor web al que deseamos llegar, www.google.es

También es conocido por todos, que lo que se mueve por *Internet*, se hace mediante **direcciones IP** (Ipv4 ó Ipv6, que son una serie de números) *y en el nombre* www.google.es *no veo absolutamente ningún número, ni nada que se parezca a una IP.*

Por tanto, si la IP (o una de ellas) de www.google.es fuese 216.239.59.104, hay "*alguien*" que se encarga de "***convertir***" **esa IP al nombre de dominio asociado o viceversa, pues ese es el cometido principal de un Servidor DNS,** *es más sencillo recordar que Vic_Thor tiene un audi de color negro en la forma* ***Vic_Thor.audi.negro***, *que el número de bastidor, ¿no?*

Cuando 2 equipos se quieren comunicar y el equipo que inicia la comunicación (equipo 1) sólo se conoce el **nombre de dominio** del otro equipo (equipo 2), el equipo 1 le pedirá a su Servidor DNS que le entregue la IP correspondiente del propietario de ese nombre (que le entregue la IP del equipo 2). Si el DNS lo sabe se lo dará, si no lo sabe podrá preguntar a otros Servidores DNS con la esperanza de que alguno lo sepa, si es así le responderá al equipo 1 y si no es así le dirá que no conoce ese nombre y no habrá comunicación con el equipo 2.

Esto es a grandes rasgos, habría que explicar más cosas del funcionamiento y comunicación entre Servidores DNS, como por ejemplo la recursividad, las iteraciones, las actualizaciones de zonas (seguras y no seguras), el registro de nombres con conflicto, sin conflicto, etc...

Algo diremos cuando llegue el momento... por ahora pensemos que el Servidor DNS es una especie de "*Dios*" que nos da la IP de un nombre de dominio, o al revés, le damos una IP y nos responde con el nombre de dominio.

Para que **los clientes DNS y Servidores DNS** puedan compartir esa información, **utilizan el protocolo DNS, que a su vez es transportado mediante otro protocolo, UDP mediante el puerto 53.**

Hay excepciones, existen casos que los Servidores DNS para actualizar sus zonas utilizan el puerto TCP 53... *No importa, olvídalo....*

Dentro del datagrama UDP, se envía la consulta DNS, que dispone de muchos parámetros, campos, flags, etc... (*Aquí es*

donde hay que repasar el formato de paquete DNS que explicó muy bien PyC en la revista nº 14), para lo que nos ocupa en éste artículo, sólo es necesario recordar unas cuantas cosas:

- **El cliente abre un puerto origen dinámico y aleatorio**, en Windows entre el 1024 y 5000 si no se indicó lo contrario en el registro o en LiNUX un puerto alto... más allá del 32768.
- **El puerto destino será el 53.**
- **Existe un campo** dentro del *datagrama DNS/UDP* ***llamado Transaction ID***, o **identificador de transacciones**, que **es generado por el cliente** y utilizarán tanto el servidor como él mismo para interrogarse mutuamente y "*pasarse*" la información.
- Existen **otros valores** dentro del *datagrama DNS/UDP* como son si se trata de una **pregunta o una respuesta**, respuesta de **autoridad**, y por supuesto, **la resolución del nombre en su IP o viceversa.**

Hay que recordar que UDP es un protocolo "*sin conexión*", ambos suponen que el otro oirá sus peticiones o respuestas, pero ninguno de ellos tienen constancia firme de ello, ***uno pide, el otro responde pero ninguno de los dos confirman se oyeron sus preguntas y respuestas.***

Hace muchos años, los ***Transaction-ID*** eran números secuenciales, con incrementos constantes, esto era así porque nadie pensó que algún "*chico malo*" embestiría contra los Servidores DNS, era un mecanismo de seguimiento y control, no de seguridad.

Hoy en día los ***Id-Transaction*** son aleatorios, no del todo, pero bastante difícil de predecir, no obstante al ser un número de 2 bytes, los valores que pueden tomar van del 0 al 65535, lo cual simplifica un poco más el asunto.

El puerto origen que abre el cliente tampoco lo sabremos a priori, ni tampoco por cual empiezan o como los abren, dispondremos de "*indicios*", ***de todas formas, no va por ahí el ataque que quiero escenificar***, si el ataque fuese contra servidores públicos... entonces sí que sería necesario, si consiguiésemos eso, casi, casi, me atrevo a decir que el servidor es nuestro.

Por el momento sabemos

- Cuando el cliente hace una petición DNS:
  - Abre un puerto dinámico.
  - Genera un identificador de transacción.
  - Espera que el servidor le responda por el puerto dinámico que abrió y con el mismo identificador de transacción,
    - Si no es así, el cliente no aceptará lo que le diga el servidor.
- Las respuestas que ofrece el servidor:
  - Incluirán una serie de nombres y/o IP's de la consulta que le hicieron.
  - En caso de que no haya podido resolver el nombre, una respuesta negativa.
  - Las entregará al Cliente por el puerto dinámico y con el mismo ID.

Y todo esto ¿qué tiene que ver, para qué lo necesitamos?

1º) Para entender un poquito más como funciona un DNS tendremos que saber cómo se pasan los datos a nivel de protocolo.

2º) Si conseguimos interceptar una respuesta DNS o una consulta DNS y cambiar los datos "*al vuelo*", le estaremos diciendo al servidor o al cliente que la IP del nombre de dominio X es la que nos de la gana y *no* aquella que verdaderamente tiene asociada y registrada ese dominio.

Si esto último lo hacemos efectivo, todos los clientes que acudan a ese servidor obtendrán unas IP's falsas y podría agravarse mucho más, pero esto ya es suficientemente grave.

Cuando un cliente de ese servidor DNS "*engañado*" le pida***... "Oye!!! Tú no tendrás la IP de www.google.com, verdad?"***

Y ese servidor DNS le diga, ***"Claro que sí, es esta, anota: 62.193.200.34"***

*Resulta que **esa IP no pertenece a google, si no a hackxcrack**, esto significa que ese cliente o todos los que le pidan resolver el nombre google.es obtendrán la IP de hackxcrack y serán dirigidos a los servidores del dominio hacxcrack.com*

El cliente se lo tragará sin rechistar, si esto lo aplicamos a nuestro amigo *"el de la imprenta"* y cuando se dirija a www.mibanco.com podemos obligarle a que aparezca en un dominio controlado por el *"atacante"*, sólo nos quedaría *"simular"* un entorno web como el de *mibanco.com. Cuando mi amigo introduzca su login y contraseña, entonces... si hicimos bien las cosas nos pegaremos una mariscadita a su salud*

***Ojito, esto no es legal**, es un delito y fuertemente penado, una cosa es "gastar una broma" o explicar en qué consiste bajo el punto de vista **educacional** dirigiendo inocentemente un equipo a google.es cuando realmente lo que quería es visitar los foros de hackxcrack y otra cosa es envenenar a miles de usuarios de un servidor DNS, más aún cuando el objeto del mismo es el robo de contraseñas y uso fraudulento de las cuentas bancarias de terceros.*

Ahora que ya sabemos a lo que vamos, de nada serviría empezar a configurar e instalar las herramientas necesarias si no entendemos cómo funciona la arquitectura del cliente DNS.

Anteriormente dije que explicaría el funcionamiento del cliente DNS en Windows, puesto que tiene alguna peculiaridad y eso es lo que haré ahora mismo...

ARQUITECTURA DEL CLIENTE DNS DE WINDOWS.

El servicio de cliente DNS hace posible que las solicitudes y consultas se empaqueten y se envíen a los servidores DNS para que éstos las resuelvan y lo harán de forma ***recursiva o iterativa.***

En Windows**, el servicio de cliente DNS **está incluido en el archivo services.exe** situado en **%SystemRoot%\System32

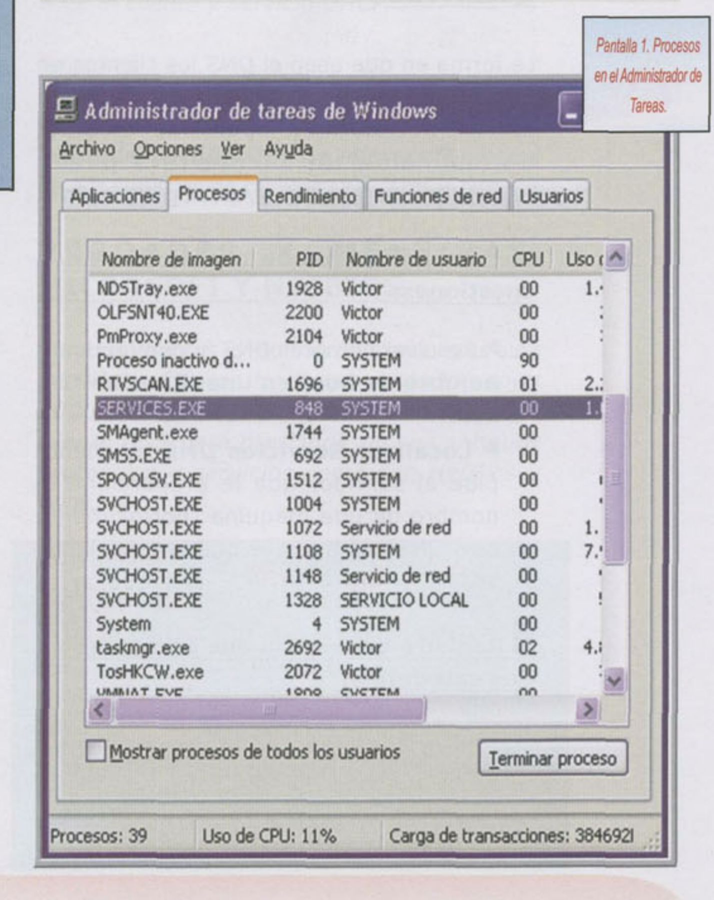
%SystemRoot% representa el directorio principal de instalación del sistema operativo, habitualmente **winnt** ó **Windows,** sencillamente sustituye por %systemroot% el directorio en el que se instaló el sistema, quedaría así: **\winnt\System32 ó \Windows\System32**

El archivo *services.exe* se ejecuta y carga en memoria cuando arranca *Windows*, lo puedes ver en el ***administrador de tareas*** (ver pantalla 1).



Para abrir...

Para abrir el Administrador de Tareas pulsar simultáneamente las teclas [Ctrl][Alt][Supr] o haz clic con el botón secundario del ratón en un espacio en blanco de la barra de tareas y, después, haz clic en Administrador de tareas.



Pantalla 1. Procesos en el Administrador de Tareas.

Este servicio **se inicia automáticamente**, **no tiene opciones de recuperación** determinadas, **ningún servicio depende de él** y **no depende de ningún otro servicio a su vez**, cuando de inicia, lo hace utilizando la cuenta ***Local System,*** bueno, en XP depende del controlador de protocolo TCP/IP y se ejecuta bajo la cuenta ***AUTHORITY\NetworkService.***

Windows 2000 (y sucesivos) es el primer Sistema Operativo de *Microsoft* que usa DNS frente a su tradicional *WINS y NetBIOS, de hecho Windows 2000 puede y debe prescindir de esos otros servicios y protocolos, con DNS y TCP/P nos basta.*

Eso es lo que a bombo y platillo dice Microsoft, después demostraremos que no es verdad y que hasta su flamante XP con SP 2 sigue siendo una trampa en lo que se refiere al DNS

COMO UTILIZAN DNS LOS CLIENTES

La forma en que usan el DNS los clientes en general y no sólo los *Windows*, depende mucho del sistema operativo elegido, **normalmente un cliente DNS puede resolver nombres y/o registrarlos.**

La Resolución puede usarse para dos cuestiones:

- Resolver nombres DNS y **convertir un nombre de host en una dirección IP.**
- **Localizar Servicios DNS**, el cliente pide al servidor que le proporcione el nombre de una máquina (normalmente otro DNS) para que le pueda consultar a ella.

El Registro también puede agruparse en dos categorías:

- **Registro de nombres** que permite que un cliente se registre dinámicamente en el servidor mediante *registros A y punteros PTR en la base de datos.*
- **Registro de Servicios**, que también son dinámicos y *añaden registros SRV* que permitirán localizar servicios particulares y otras cuestiones como el coste, el peso, etc...

No te asustes con esas siglas, cuando veas (o leas) registros A, MX, PTR, etc... Recuerda esta tabla

Registro	Descripción
A	Máquina o host individual, un servidor web, un equipo cualquiera
MX	Servidor de correo
PTR	Puntero o entrada inversa, en lugar de la correspondencia nombre a IP es lo contrario, IP a nombre
SRV	Localizador de recursos
NS	Se trata de un servidor DNS

Para todo lo que se refiere a este artículo, con saber diferenciar entre ***Registros A y registros NS es suficiente.***

Otro factor importante a la hora de registrar un cliente en un DNS es **utilizar un nombre cualificado, lo que se llama *FQDN o Full Quality Domain Name.***

El nombre del host (hay quien le llama anfitrión) **es un "*objeto*" una parte del FQDN**, **el otro componente es el sufijo de dominio.**

El nombre identifica el host, **el sufijo se utiliza para cualificar ese nombre de anfitrión**, *así por ejemplo si mi máquina se llama portatil-VIC y pertenece al dominio hackxcrack.com, mi FQDN sería portatil-VIC.hackxcrack.com.* ***Y ese último punto al final es importante, con el terminamos el FQDN para esa máquina.***

Los nombres de anfitrión deben estar compuestos como máximo por 25 caracteres y sólo pueden usarse letras mayúsculas o minúsculas, números del 0 al 9 y el signo menos (-)

Ni que decir tiene que no vale la Ñ ó ñ, ni vocales acentuadas, ni nada de eso... solo letras, números y/o signos menos.

En cuanto a los sufijos.... tenemos tres tipos:

1. **Sufijos DNS principal**, sólo puede haber un principal por cada máquina y permite realizar consultas cualificadas o no al servidor DNS.

2. **Sufijos específicos de conexión**, permiten asignar sufijos "*alternos*" en cada tarjeta de red del host, de ese modo un mismo host puede registrarse en varios dominios y/o DNS, *esto es muy útil cuando configuramos un cortafuegos, un servidor de acceso remoto o un servidor de enrutamiento, cada tarjeta de red pertenece a un espacio de nombres diferente en una misma máquina*

3. **Sufijos DNS adjuntos** se usan para la resolución de nombres, como los principales, sólo que estos permiten que una consulta no cualificada se convierta en cualificada para ese espacio de nombres definido por los adjuntos, pueden existir tantos sufijos adjuntos como nos de la gana, pero no por poner muchos la cosa irá mejor...

Ahhh!!! Claro.... ¿Y qué demonios es una consulta cualificada?

Es la consulta a un DNS de un nombre FQDN, si todos los nombres y búsquedas son cualificadas, el rendimiento el la red y la eficiencia de los servidores DNS serán mucho mayor que si usamos consultas no cualificadas, éstas últimas precisan de un montón de preguntas y respuestas para resolver ese nombre, se moverá más tráfico por la red, consumiremos ancho de banda y nuestras consultas se resolverán en más tiempo.

Servicios DNS del lado del cliente.

- **DNR**, *Domain name Resolver* o **Servicio de resolución de nombres.**
- **CRS**, *Caché Resolver Service*, **Servicio de resolución de nombres en caché.**
- **DCS**, *DNS Client Service*, **Servicio de Cliente DNS.**

Son términos de Microsoft y los podrás encontrar por muchos libros, con diferentes definiciones, funciones, descripciones. Para nosotros y para este artículo, los tres serán uno sólo (como La Trinidad) ***DNR+CRS+DCS = Servicio Cliente DNS*** a partir de ahora.

El servicio Cliente DNS, **es el responsable de enviar las consultas DNS a un servidor y de recibirlas**, cuando una aplicación, un proceso, u otro servicio precisa que este servicio sea invocado, lo primero que hará es ***llamar a una API de nombre GetHostByName ó GetHostName (ahora no recuerdo)***, y se encargará de poner en marcha el proceso de resolución de nombre, tanto si se tiene éxito como si no, el resultado se entrega al solicitante.

El servicio de cliente DNS se carga dentro del archivo services.exe (es una parte del mismo, no todo) y se inicia con privilegios de System, ya.... ya lo has dicho antes, ¿es que no te acuerdas?

Sí, claro que lo recuerdo, pero lo repito de nuevo por lo siguiente, el archivo ***services.exe*** *"guarda en su interior" alguna sorpresa, entre ellas el cliente y servicio DHCP*

Ya... ¿Y?

Pues bien... vamos a contar algunas de las **verdades y mentiras de *Microsoft*** acerca de sus servicios de clientes DNS, DHCP, NetBios

VERDADES Y MENTIRAS, MICROSOFT Y NETBIOS

Como dije antes (y no son palabras mías sino de *Microsoft*) *a partir de la versión Windows 2000, Microsoft implementa el servicio de cliente DNS para prescindir de sus antiguos protocolos y servicios como son NetBios y Wins.*

Por si no lo sabes*: tanto NetBIOS como Wins son una forma de comunicar equipos Windows entre ellos o con el mundo exterior y aunque el responsable en el desarrollo de NetBIOS no fue Microsoft, es la arquitectura por la que ha optado durante muchos años. Ahora Microsoft dice que su Sistema Operativo ya no requiere de NeTBIOS, es más, hasta incluye opciones para deshabilitar NetBIOS.*

El problema de usar o no ***NetBIOS*** es que si solamente usamos ese y ningún otro, sólo podremos comunicarnos con equipos *Windows* que estén compartiendo los medios de conexión con nosotros, es decir, ni podremos salir a Internet, ni comunicarnos con otras redes puesto que no es un protocolo enrutable pensado para la filosofía de "*trabajo en grupo*"

Si añadimos *TCP/IP a NetBIOS* en la lista de protocolos a usar, podremos comunicarnos con otros equipos, otras redes y por supuesto con Internet, puesto que TCP/IP nos brindará un conjunto de protocolos para ello, eso es lo que a veces se llama la pila TCP/IP.

La pregunta es obvia... *¿Si TCP/IP hace lo mismo que NetBIOS y además tiene más ventajas, para qué usar NetBIOS junto con TCP/IP?*

¿Lógico no? *Es como si tuviésemos dos procesadores de texto, uno que sólo escribe en mayúsculas y otro que permite escribir tanto en mayúsculas como en minúsculas... pues para qué instalar los dos, con el último de ellos cubrimos las dos necesidades.*

El caso es que por "*herencia*" *NetBIOS* se sigue manteniendo en las últimas versiones de *Windows* y aunque dispongamos de órdenes o pantallitas para poder deshabilitar NetBIOS, yo voy a demostrar que no es así, es más... que ***Microsoft* miente al decir que su "*Windows 2000*" no depende de NetBIOS** y que escala perfectamente en arquitecturas DNS y TCP/IP, MENTIRA.

*¿**Y aunque demuestres eso, qué tiene que ver con lo que nos ocupa**?*

Pues muy simple, ***resulta que tanto DHCP como DNS* dependen de NetBIOS** (aunque Microsoft siga diciendo que no), es más, si alguno se piensa que por disponer de una IP asignada por el administrador (fija) no está utilizando el servicio DHCP, se equivoca. Aunque creamos que el servicio DHCP se utiliza sólo cuando no tenemos IP fija, eso no es cierto!!! **El servicio DHCP se utiliza SIEMPRE en un Windows 2000, no importa si tienes IP fija, automática o si no la tienes, y se utiliza para registrar ese nombre en el DNS.**

Resulta que tanto *DHCP y DNS* se encuentran dentro de ese *misterioso* archivo ***services.exe* que se ejecuta SIEMPRE** queramos o no... pero lo peor no es eso... es que aunque Microsoft diga que su "Windows 2000" está libre de NetBIOS, ES MENTIRA, como lo del **DNS y el DHCP, estos servicios siguen dependiendo de *NetBIOS* y la estabilidad del sistema y del acceso a otras redes TAMBIÉN DEPENDE DE NETBIOS, aunque ellos digan que no...**

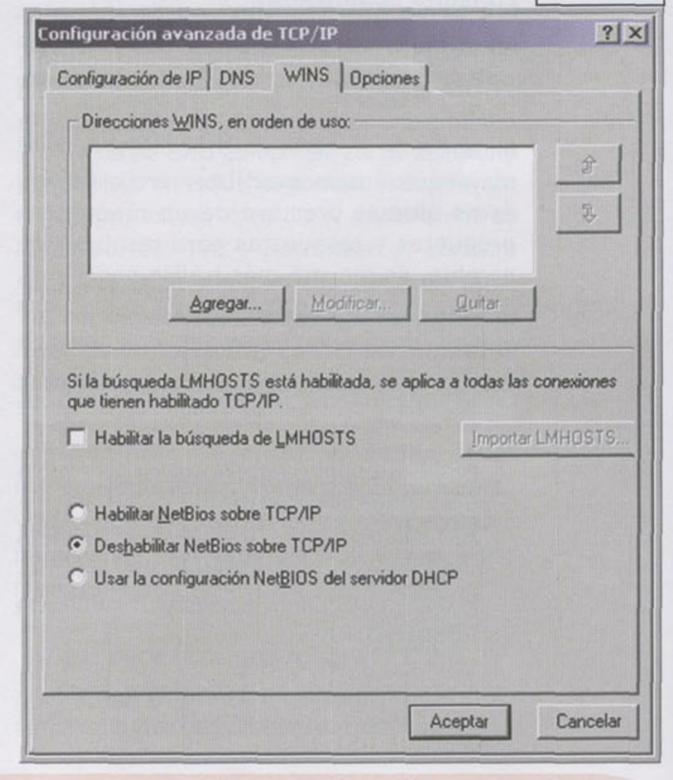
Todo esto lo digo para que vayamos haciéndonos a la idea que el cliente del servicio DNS de Microsoft es "*peculiar*", como también recordarás que apunté anteriormente.

Antes de analizar el cliente DNS de *Microsoft*, veamos esas pruebas acerca de *NetBIOS*

Microsoft dice que **para deshabilitar NetBios tenemos que hacer lo siguiente:**

1. Abrimos ***Propiedades*** de la Tarjeta de red.
2. En la ***ficha General***, elegimos ***protocolo TCP/IP.***
3. ***Pulsamos en Propiedades.***
4. ***De nuevo en el botón de Avanzadas*** dentro de la pantalla de Propiedades
5. Pulsamos en la ***ficha WINS.***
6. ***Seleccionamos Deshabilitar NetBIOS sobre TCP/IP*** (ver pantalla 2).

Pantalla 2. Cómo deshabilitar NetBios sobre TCP/IP



Y luego vamos pulsando en ***Aceptar*** hasta que ya no aparezcan más cuadros de dialogo 😊

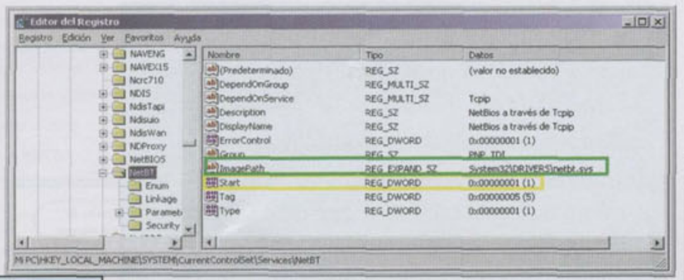
Si disponemos de más de una Tarjeta de Red*, hay que hacerlo individualmente por cada una de ellas, y se supone que ya nos libramos de NetBIOS.*

Mentira cochina.... veamos la primera prueba.

Si accedemos al registro (***Inicio-Ejecutar-regedit***) y buscamos la clave:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ServicesNetBT

Como muestra la pantalla 3, nos daremos cuenta que la clave ***Start*** (*recuadrada en amarillo*) tiene un valor de uno (1) y eso significa que cuando *Windows* arranque o reinicie la siguiente ocasión, cargará el archivo ***netbt.sys*** tal y como apunta el valor ***ImagePath*** (*en verde*).



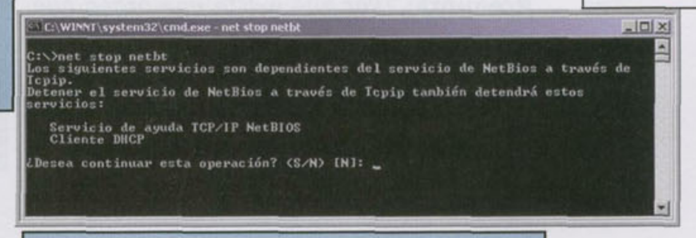
Pantalla 3.- Valores del Registro de Windows con NetBIOS desactivado....

Ese valor le indica al núcleo del Sistema Operativo que debe cargar ese controlador como componente básico del sistema... como el teclado o el controlador de vídeo... vamos que aunque hayamos deshabilitado *NetBIOS*, sigue cargándolo y ***ojito... si nos da por modificar el valor de Start en la pantalla anterior y ponerlo a cero (que no cargue el controlador) WINDOWS NO ARRANCARÁ!!!! Así que no lo hagáis, no toquéis esos valores del registro que os cargáis vuestro bonito Windows.***

Bien, *podemos pensar* que el mero hecho de tener un controlador cargado en memoria como parte del núcleo del sistema no obliga a usarlo, es decir, que podemos tener cargado *NetBIOS* y no usarlo**... ja.... eso es lo que pensamos todos...**

Microsoft, amablemente nos entrega una orden para detener el servicio NetBIOS, la manera mas sencilla es ***Inicio ---> Ejecutar ---> cmd*** y desde la consola iniciar el comando ***net stop netbt***

Pantalla 4.- Detener el Servicio NetBIOS



AYYYYYY!!! Resulta que cuando intentamos parar NetBIOS nos dice que si lo hacemos se detendrán también el Cliente DHCP y el Servicio de ayuda TCP/IP NetBIOS.

Pregunta: *¿No le habíamos dicho antes que deshabilitara NetBIOS bajo TCP/IP?*

Pregunta: *¿Qué tiene que ver DHCP con NetBIOS?*

Pero es que hay más... **si eso mismo lo hiciésemos en un Servidor DNS**, resulta que además de detener esos dos servicios **detendremos el propio servicio DNS.**

Pantalla 5. Detener el servicio NetBIOS en un Servidor DNS.

```
Seleccionar C:\WINNT\system32\cmd.exe - net stop netbt
C:\>net stop netbt
Los siguientes servicios son dependientes del servicio de NetBios a través de
Tcpip.
Detener el servicio de NetBios a través de Tcpip también detendrá estos
servicios:

   Servicio de ayuda TCP/IP NetBIOS
   Servidor DNS
   Cliente DHCP

¿Desea continuar esta operación? (S/N) [N]:
```

Pregunta: *¿No decía Microsoft que DNS es independiente de NetBios?*

Pregunta: *¿Por qué al detener NetBIOS se detiene DNS si no dependen uno del otro?*

Hay más... **vamos a decirle que S (Sí)** y observa....

Pantalla 6. Deteniendo el Servicio NetBIOS en una máquina

```
C:\WINNT\system32\cmd.exe
C:\>net stop netbt
Los siguientes servicios son dependientes del servicio de NetBios a través de
Tcpip.
Detener el servicio de NetBios a través de Tcpip también detendrá estos
servicios:

   Servicio de ayuda TCP/IP NetBIOS
   Cliente DHCP

¿Desea continuar esta operación? (S/N) [N]: s
El servicio de Servicio de ayuda TCP/IP NetBIOS está deteniéndose.
El servicio de Servicio de ayuda TCP/IP NetBIOS fue detenido con éxito.

El servicio de Cliente DHCP está deteniéndose.
El servicio de Cliente DHCP fue detenido con éxito.

El servicio de NetBios a través de Tcpip está deteniéndose........
El servicio de NetBios a través de Tcpip no ha podido detenerse.

C:\>
```

Dice que el NetBios está deteniéndose.... y **luego nos informa que NO puede detenerse***.... eso sí, el DHCP nos lo hemos cargado....*

Pregunta: *¿Por qué existe una orden que detiene un servicio si luego ese servicio no puede ser detenido y en su lugar detiene otros que no le he dicho que detenga?*

Pregunta: *Si no puede detener NetBIOS... ¿Por qué no deja como estaba los otros?*

Bueno... vamos a pensar que "*algo pasó*".... ***repetiremos el comando, a ver si a la segunda....***



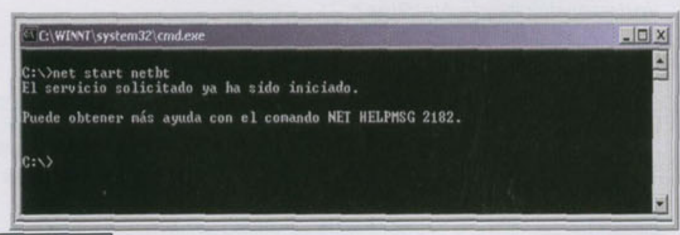
Pantalla 7. Deteniendo el servicio NeTBIOS por segunda ocasión...

Resalto en amarillo esa frase... es antológica... "*No se puede controlar el servicio en su estado actual*".

Pregunta: *¿Controlar? ¿Pero no soy yo quien "controlo" si deseo usar NetBIOS o no? Define Controlar....*

Pregunta: *¿Controlar? ¿Es que se ha desmadrado o se salió la tarjeta de red del slot?*

Ufff, venga, no pasa nada... ***lo habilitamos de nuevo y ya está... net start netbt***



Pantalla 8. Iniciar el Servicio NetBIOS "detenido" previamente

CIELOS!!!! Ya está iniciado!!!

Pregunta: *¿No hemos deshabilitado el servicio desde las ficha WINS?*

Pregunta: *¿No decía que no se podía controlar?*

Y no voy a seguir, porque si después de todas estas inconsistencias, inicios, detenciones de servicio etc... Resulta que **colocamos un esnifer a escuchar el tráfico NetBIOS y/o SMB, SALE Y ENTRA de la tarjeta de red... o sea, que nada de nada....**

RESPUESTAS A TODAS ESAS PREGUNTAS.

1. **FELICIDADES**, si has seguido todos los pasos **acabas de desestabilizar por completo a tu Windows,** bienvenido al acceso a una red imprevisible, de comportamientos inesperados y maravillas de la conectividad, sólo te queda una opción: **REINICIAR EL SISTEMA!!!**

2. **NO SE PUEDE detener NetBIOS**, si lo haces todo deja de funcionar, luego todo (o casi todo) depende de *NetBIOS* aunque *Microsoft* diga que Windows 2000 es independiente de *NetBIOS*.

Si hacemos un paralelismo *no deja de "acongojar". Pongamos que somos el piloto de un avión... y además de todos los cachivaches, palancas, mandos, lucecitas, etc.. Tenemos dos botones: Uno que dice Accionar el piloto automático y otro que dice Desactivar piloto automático.*

*Cuando el avión despega, lo hace con el piloto automático activado (como Windows con NetBIOS) y si un buen día el comandante del avión decide pulsar el botón de Desactivar el piloto automático... el avioncito le responde que no puede, lo intenta por segunda vez y le dice que el piloto automático está descontrolado.... joer, no me digas que no... Debe estar "cagado" pensando qué ocurrirá cuando pulse el botón de Sacar el tren de aterrizaje, o el freno... si lo pulsa una vez, no puede y si lo hace dos... se descontrola****.... FELIZ VUELO!!!***

Ahh!!! Otra cosa... en todo este artículo estoy haciendo referencia a *Windows 2000,,,* ***y de Windows XP?***

Pues **más de lo mismo...** no lo voy a repetir, pero mira lo que pasa sobre un *Windows XP* con su flamante *Service Pack 2* al *"intentar"* deshabilitar *NetBIOS* como hemos explicado ... cuando se intenta acceder a la red...

Mis sitios de red

No tiene acceso a \\172.28.0.254\wwwroot. Puede que no tenga permiso para utilizar este recurso de red. Póngase en contacto con el administrador de este servidor para comprobar si tiene permisos de acceso.

No es posible el acceso a la ubicación de red. Para obtener información para solucionar problemas de red, vea la Ayuda de Windows.

Aceptar

Pantalla 9. Acceder a una red sin NetBIOS pero con TCP/IP activado....

ZAS!!! O reiniciamos la máquina o nos quedamos sin red....

Regresemos a tierra.... sigamos con nuestro cliente DNS.

Todo esto venía a cuento de cómo el Cliente de Microsoft actúa y procesa la información para conseguir relacionar un nombre con su IP correspondiente y aunque efectivamente utiliza el servicio DNS, podemos realizar la misma técnica de ataque (cuando llegue, calma) por NetBIOS... AUNQUE LO DESACTIVEMOS COMO DICE MICROSOFT!!!!

PROCESOS DE UNA CONSULTA DNS DE UN CLIENTE WINDOWS

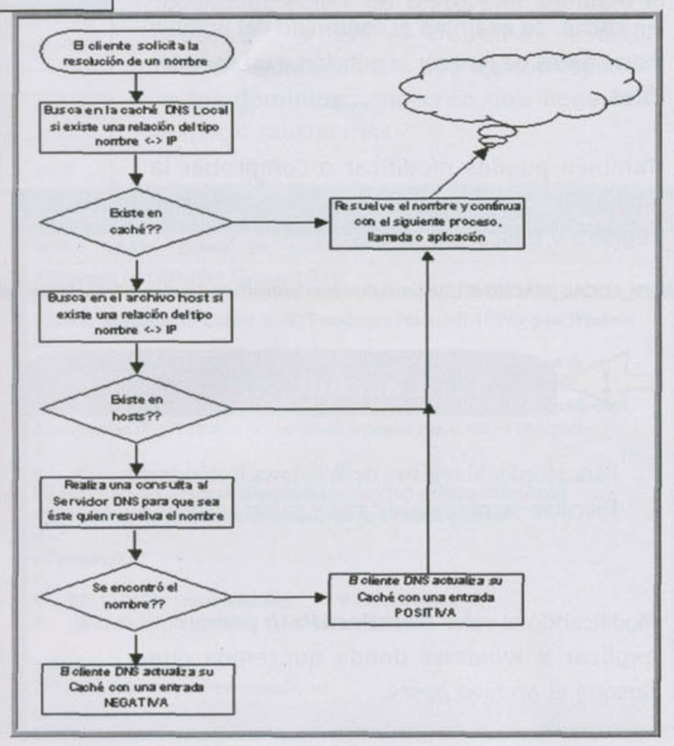


Figura 1. Pasos para resolver un nombre por un cliente Windows

Lo primero es pedir perdón por el organigrama.... los puristas en el diseño de los mismos se estarán tirando de los pelos, ya sabes, de arriba abajo y de izquierda a derecha... y en el mío... bueno, creo que se entiende, ¿no? Explicaremos un poquito más.

1. Cuando un cliente DNS precisa resolver un nombre o que se lo resuelvan, lo primero que **hace es una llamada a la *API GetHostByName*** mediante el archivo ***services.exe*** que es que ejecuta el cliente. Una vez procesada la información s**e crea un paquete TCP/IP con los formatos de datagrama DNS/UDP** solicitando que le sea entregada la IP que corresponde al nombre de dominio a buscar.

2. Se comprueba si existe una entrada válida (positiva) en la caché del cliente y si existe se resuelve el nombre por su IP, si no existe se pasa al punto siguiente.

3. Se comprueba si existe el archivo HOST, normalmente está situado en ***%systemroot%\system32\drivers\etc.***

4. Realiza una petición al servidor DNS para que le resuelvan el nombre y su IP.

Cuando no existe entrada previa en la caché y tampoco en el archivo *hosts*, llegamos a este paso.

En este proceso se envían un máximo de 5 consultas DNS a cada servidor DNS que el cliente tenga configurado en el protocolo TCP/IP, en estricto orden secuencial, es decir, si la primera consulta al primer servidor es positiva, se resuelve el nombre correctamente y se actualiza la caché del cliente DNS con una entrada positiva, si a la primera no lo consigue lo intenta cuatro veces más y si tampoco lo logra, hace lo mismo con el siguiente servidor DNS si lo hay... así hasta agotar los **cinco intentos por cada servidor DNS asignado al cliente.**

¿QUÉ ES LA CACHÉ DNS?

A partir de *Windows 2000*, *Microsoft* incorpora una "*técnica*" que **se trata de almacenar en memoria aquellas resoluciones de nombre positivas o negativas de las consultas que se han efectuado.**

Una entrada caché positiva es aquélla que **se pudo resolver mediante un servidor DNS o mediante el archivo *hosts*** (que veremos más adelante).

Una entrada negativa es aquella que **NO se pudo resolver** porque ninguno de los servidores DNS a los que se consultaron conoce la IP del nombre de dominio que se busca.

Por defecto **la caché del cliente DNS puede mantener unas 200 líneas** (211 concretamente) **y 10 columnas**, dependiendo del número de caracteres de cada línea y de sus columnas, tendremos más o menos nombres cacheados.

Ciertamente **el uso de la memoria caché** en los clientes DNS puede parecer un avance y en condiciones normales lo es, puesto que si el equipo ya conoce la IP de un dominio determinado no tiene por qué volver a realizar una petición DNS para resolverla, de ese modo se **reduce el tráfico por la red, se dispone de un mayor ancho de banda y se tarda menos tiempo en las conexiones.**

Por el contrario, si la caché se corrompe o se incluyen entradas falsas, tendremos un equipo "*loco*" que no es capaz de resolver nombres o que se comporta de forma extraña.

***En este artículo** terminaremos por envenenar la caché de un cliente DNS (o de todos) que pertenece a una red. De esta forma, cuando quiera "navegar" a una determinada dirección, o bien le provocamos un DoS (negación del servicio) o bien le redirigimos a otro sitio que no es el que realmente debería ser.*

Cuando una caché es envenenada o corrompida, hay que vaciarla o reiniciar el equipo (esta es una solución más drástica pero igual de efectiva). El caso es que si nos lo montamos bien, dispondremos de mucho tiempo (un día máximo) para obligar a ese cliente a que haga lo que nosotros queramos y no lo que el usuario que está frente al teclado quiere que haga... *esta es la esencia de este artículo.*

Más adelante veremos algunos comandos útiles y entradas en el registro de *Windows* (***regedit.exe***) que nos permiten consultar la caché DNS y/o modificar su comportamiento.

EL ARCHIVO hosts

El archivo hosts lo puedes encontrar en:

%systemroot%\system32\drivers\etc

Es un archivo que contiene una asignación manual (administrativa por parte del usuario) que relaciona nombres de dominio con sus IP's de tal forma que después de comprobar que el nombre de dominio a buscar no existe en caché, se examina el contenido del archivo hosts antes de realizar la petición a un servidor DNS.

También puedes modificar o comprobar la ubicación de este archivo si accedemos al registro y buscamos la clave:

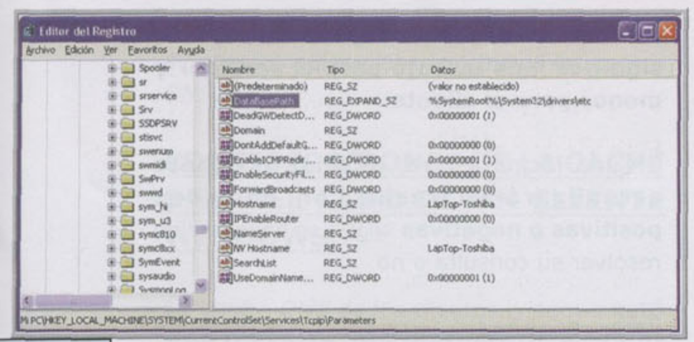
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters



Para acceder...

Para acceder al registro de Windows Botón Inicio ---> Ejecutar, escribir regedit.exe y pulsar aceptar.

Modificando el valor ***DataBasePath*** podremos explicar a *Windows* donde queremos que busque el archivo *hosts*.



Pantalla 10. Ubicación del archivo host mediante valores del Registro.

ADVIERTO: *Si modificas la ubicación del archivo hosts, tendrás que ubicar en la misma ruta los archivos: LMHOST.sam, Services, Networks y Protocol, es decir, que la entrada del registro se aplica a todos esos archivos y no sólo a hosts*

```
C:\WINDOWS\system32\cmd.exe
^C
C:\Documents and Settings\Victor>cd\windows\system32\drivers\etc

C:\WINDOWS\system32\drivers\etc>dir
 El volumen de la unidad C no tiene etiqueta.
 El número de serie del volumen es: 0759-1602

 Directorio de C:\WINDOWS\system32\drivers\etc

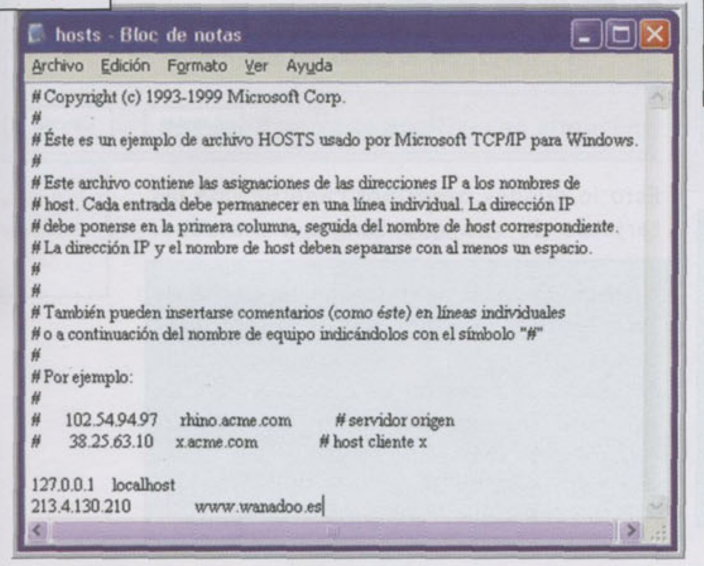
26/05/2003  15:16    <DIR>          .
26/05/2003  15:16    <DIR>          ..
10/09/2002  13:00               792 hosts
10/09/2002  13:00             3.429 lmhosts.sam
10/09/2002  13:00               418 networks
10/09/2002  13:00               908 protocol
10/09/2002  13:00             7.666 services
19/07/2004  01:49               375 hosts.ics
               6 archivos         13.588 bytes
               2 dirs  10.419.568.640 bytes libres

C:\WINDOWS\system32\drivers\etc>_
```

Pantalla 11. Archivos que dependen de la ubicación del valor DataBasePath

El archivo *hosts* se puede ver o modificar con cualquier editor de texto, por ejemplo el *notepad*, y en él podemos escribir manualmente las IP's que les corresponden a los dominios... no creo que haga falta explicar mucho más...

Pantalla 12. Edición del archivo hosts con notepad


hosts - Bloc de notas

Archivo Edición Formato Ver Ayuda

```
# Copyright (c) 1993-1999 Microsoft Corp.
#
# Éste es un ejemplo de archivo HOSTS usado por Microsoft TCP/IP para Windows.
#
# Este archivo contiene las asignaciones de las direcciones IP a los nombres de
# host. Cada entrada debe permanecer en una línea individual. La dirección IP
# debe ponerse en la primera columna, seguida del nombre de host correspondiente.
# La dirección IP y el nombre de host deben separarse con al menos un espacio.
#
#
# También pueden insertarse comentarios (como éste) en líneas individuales
# o a continuación del nombre de equipo indicándolos con el símbolo "#"
#
# Por ejemplo:
#
#     102.54.94.97     rhino.acme.com          # servidor origen
#      38.25.63.10     x.acme.com              # host cliente x

127.0.0.1       localhost
213.4.130.210          www.wanadoo.es
```

Podemos añadir las líneas que queramos, en cada una de ellas aparecerá la IP y el nombre de host a resolver. Obviamente ***este sistema es administrativamente penoso***, pues si disponemos de muchas máquinas en la red, tenemos que incluir un archivo *hosts* para cada una de ellas y por cada nuevo "*descubrimiento*" deberíamos cambiarlo por cada máquina.

Cuando nació Internet, los archivos *hosts* (algo parecido) se descargaban de servidores FTP y así se mantenían las tablas de relaciones dominio<->IP

Si los datos del archivo *hosts* son obsoletos o no se corresponden con la realidad, la máquina no resolverá bien la dirección y se producirán errores o accesos a direcciones incorrectas.

Por ejemplo, si en lugar de escribir en el archivo *hosts* la verdadera IP de www.wanadoo.es, ponemos la IP que le corresponde a www.terra.es, salvamos los cambios en el archivo, cuando el usuario de esa máquina abra su navegador y escriba la dirección www.wanadoo.es aparecerá en la página de *terra.es* 😉

Esto es lo que hace concretamente el archivo hosts mostrado anteriormente, si incluyes esa última línea en tu archivo hosts, comprobarás por ti mismo lo que ocurre cuando navegas a al web de www.wanadoo.es

Si falla la búsqueda en la caché y en el archivo hosts, es cuando el cliente solicita la resolución del nombre a sus Servidores DNS, 5 veces por servidor, como ya se explicó antes.

Además, si el cliente dispone de varios sufijos de dominio, los va anexando a su consulta para intentar registrarse mediante las consultas no cualificadas o cualificadas. Lo que dijimos unas líneas atrás, el caso es que es un proceso largo de explicar y afortunadamente corto en el tiempo.

COMO TRATA LAS CONSULTAS EL SERVIDOR DNS

Cuando un servidor DNS recibe una consulta de un cliente DNS, puede ocurrir dos cosas:

A.- **Que ya conozca la solución** (**la tiene en su caché DNS Server**) y se la entrega al cliente que la solicitó

B.- **Que no la conozca**, entonces el servidor DNS puede:

B.1.- **Ponerse en contacto con otros Servidores DNS** y preguntarles (**modo recursivo)**

B.2.- **Entregar al cliente una lista de Servidores DNS** para que sea el cliente quien les pregunte directamente (**modo iterativo**)

***Esto es parecido** a cuando llamas a Información telefónica para pedir el número de teléfono de alguien, les indicas el nombre de "ese alguien" (nombre de host) y también le dices la ciudad o incluso la calle (sufijo de dominio), si la operadora operase en **modo recursivo**, iría preguntando "por ahí" el número de teléfono de alguien.calle.ciudad y si lo encuentra te lo da.*

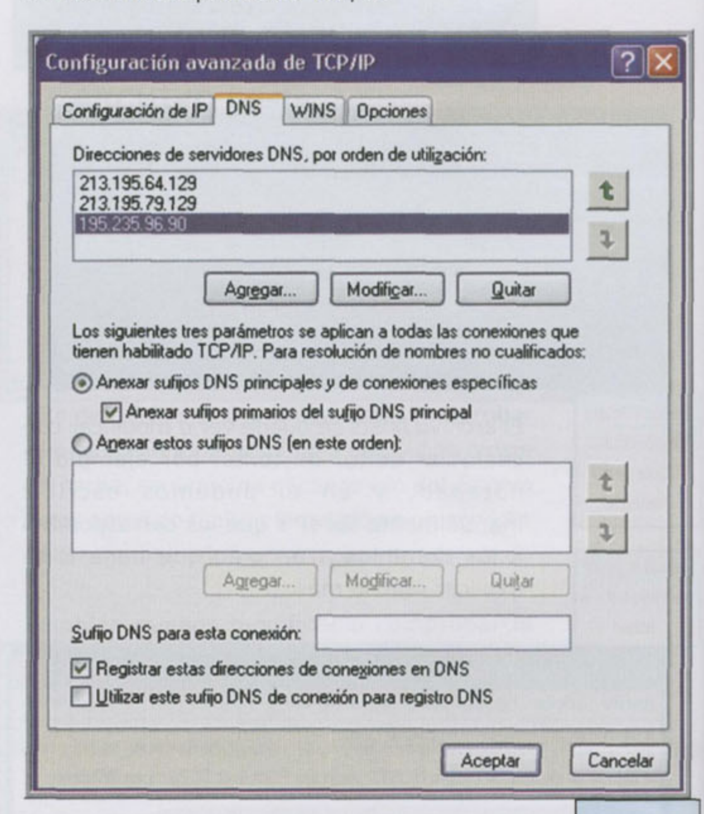
*Si el servicio de información operase en **modo iterativo**, te daría una lista de otros operadores del servicio de información telefónica para que seas tú quien los llame preguntando si saben el teléfono de alguien.calle.ciudad*

Lo más habitual es que las consultas sean recursivas, puesto que de ese modo se consume menos ancho de banda en la red. Como consecuencia negativa está que el servidor DNS "**trabaja más**", excepto e el caso de que ya tuviese almacenada esa consulta (caché DNS), que suministraría la información inmediatamente.

Podríamos decir que **un servidor DNS que opera *recursivamente* es muy parecido a un proxy**, representa a su cliente ante otros servidores y lo hace en su nombre, **esto significa más trabajo para el servidor y menos para el cliente**.

De una forma u otra, **al final el cliente actualizará su caché con entradas positivas o negativas** según se haya podido resolver su consulta o no.

Para ver, añadir o modificar, la lista de servidores DNS disponibles por el lado del cliente y para añadir sufijos (principal, alternativo o adicionales) puedes acceder a la ficha DNS de las propiedades de la tarjeta de red dentro del protocolo TCP/IP.



Pantalla 13. Ficha DNS del protocolo TCP/IP

Esto lo puedes hacer por cada una de las tarjetas de red que tengas instaladas.

***Nosotros vamos a envenenar la caché de un cliente o servidor DNS con entradas falsas**, entradas de nombres de dominio que apuntan a IP's que no les corresponden. De ese modo cuando un cliente las consulte recibirá unos valore que no son reales. Si nos esmeramos y conseguimos el objetivo, tendremos la red en nuestros "dedos".*

Antes de empezar al tajo, hay que observar algunas consideraciones acerca de las *caches DNS*...

CONSIDERACIONES DE LA CACHÉ DNS DEL LADO DE UN CLIENTE WINDOWS.

La caché DNS de los clientes *Windows* **está HABILITADA por defecto** y el **valor predeterminado de la misma es de 86.400**

Ese número **son los segundos que una entrada positiva permanece en caché...** 86.400 segundos es **UN DÍA!!!**
24 horas x 60 minutos x 60segundos = 86.400segundos.

Esto es importante... un valor inferior a 86.400 DESHABILITA LA CACHÉ!!!!

No obstante, cuando la consulta es resuelta por un servidor DNS, éste también indica el tiempo de vida (***TTL***) de esa resolución en caché, o sea, que **si un cliente pide que le resuelvan un nombre de dominio y el servidor DNS le coloca un tiempo de vida mayor, se queda con ese valor... y si le envía uno menor, se queda con 86.400.**

Teóricamente no hay límite para el número de entradas (positivas y negativas) pero también **de forma predeterminada se resumen en unas 211 filas por 10 columnas de texto.**

Por cada entrada positiva, se almacena

- El nombre del dominio
- La IP que le corresponde
- El tiempo de vida
- El tipo de registro que se trata (A, NS, MX, SRV, PTR, etc...), l
- La forma en el que el cliente recuperará la asignación (pregunta, archivo host, resolución inversa, etc..)
- La longitud de los datos en caché (lo que ocupa el nombre)

Las *caches* negativas o entradas negativas

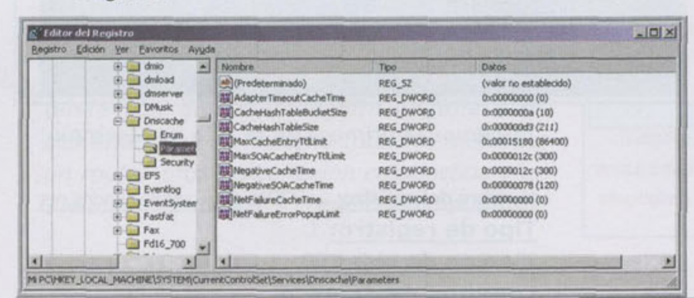
- Tienen un **valor predeterminado de existencia de 300 segundos** (5 minutos)
- También puede manipular el servidor DNS el ***TTL***, pero **NUNCA excederá de 20 minutos (1.200 segundos)**
- Las entradas negativas almacenan menos información, los únicos datos que guarda son:
  - El nombre ***FQDN*** de la búsqueda fallida.
  - El tiempo de vida (***TTL***)
  - **Un indicador** que especifica que se trata de una caché negativa.

Todos estos valores se pueden ajustar a voluntad *"tocando"* el registro de *Windows*, **las claves más interesantes y sus valores son:**

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\DnsCache\Parameters

Y las subclaves serían:

- ***MaxCacheEntryTtlLimit*** con un valor de 86.400 para el tiempo de vida en caché positiva
- ***CacheHashTableBucketSize*** con un valor de 10 para el número de columnas
- ***CacheHashTableSize*** con un valor de 211 para el número de filas en caché
- ***NegativeCacheTime*** con un valor de 300 para el tiempo de vide en caché negativa



Pantalla 14. Valores significativos de las caches desde el Registro.

Hay mas valores interesantes y no sólo en esta sección. Si visitamos las subclaves y valores para cada interface hay muchas cosas que podemos modificar... precisaríamos de un artículo entero para "indagar" algo más en el registro de Windows, o de una serie de artículos llamada "Registro de Windows"... a ver si alguien se anima que me veo trabajando en ello.

Desde el registro de Windows podemos cambiar muchos aspectos de su comportamiento, desde hacerlo inservible hasta convertirlo en un espía invisible pasando por cambios de apariencia o modificarlo para que sea el troyano que más líneas de código utilizó jamás.... o para que nos envíe lo que queramos a una máquina remota, vamos que hay tela marinera...

Desde la línea de comandos podemos realizar también algunas operaciones con la caché DNS del cliente,

Ipconfig/displaydns que **mostrará el contenido de la caché** en ese momento:

```
C:\WINNT\system32\cmd.exe
    Registro CNAME. . . . :
                    www.google.akadns.net

    Nombre de registro. . : www.google.akadns.net
    Tipo de registro . . .: 1
    Tiempo de vida  . . . : 50
    Longitud de los datos : 4
    Sección . . . . . . . : Answer
    Un registro (Host). . :
                    66.102.11.99

    Nombre de registro. . : www.google.akadns.net
    Tipo de registro . . .: 1
    Tiempo de vida  . . . : 50
    Longitud de los datos : 4
    Sección . . . . . . . : Answer
    Un registro (Host). . :
                    66.102.11.104

  27.161.169.213.in-addr.arpa.
  ----------------------------------------
    Nombre de registro. . : 27.161.169.213.in-addr.arpa
    Tipo de registro . . .: 12
    Tiempo de vida  . . . : 56701
    Longitud de los datos : 4
    Sección . . . . . . . : Answer
    Registro PTR. . . . . :
                    ip-161-27.evhr.net

    Nombre de registro. . : 161.169.213.in-addr.arpa
    Tipo de registro . . .: 2
    Tiempo de vida  . . . : 56701
    Longitud de los datos : 4
    Sección . . . . . . . : Authority
    Registro NS . . . . . :
                    dns1.sdv.fr

    Nombre de registro. . : 161.169.213.in-addr.arpa
    Tipo de registro . . .: 2
    Tiempo de vida  . . . : 56701
    Longitud de los datos : 4
    Sección . . . . . . . : Authority
    Registro NS . . . . . :
                    dns2.sdv.fr
```

Pantalla 15.- Resultado de la orden ipconfig/displydns

Tomemos la primera de la lista por ejemplo:

Nombre del registro: su FQDN www.google.akadns.net
Tipo de registro: 1
Tiempo de vida: 50
Longitud de datos: 4
Sección: Answer
Un registro Host: 66.102.11.99

El ***FQDN*** está claro. Igual ocurre con el ***tiempo de vida que son 50 segundos*** pero como el valor predeterminado de existencia en caché es de 86.400, serán estos y no los 50 segundos lo que "*sobreviva*" la entrada, recuerda que se queda con el valor más grande.

La longitud de 4 son los bytes que ocupa en memoria. **La sección es el modo en el que el cliente resolvió la dirección IP**, como pone *Answer* quiere decir que fue mediante una consulta a un servidor DNS y **la IP... pues eso... lo que se buscaba**...

Lo que no queda tan claro es lo del **Tipo de registro**. Dijimos que podrían ser (entre otros) *A, MX, PTR, SRV....* y en lugar de eso, ***nos coloca un uno (1)***

Esto es porque **los clientes DNS Windows utilizan otra forma** de llamar a los registros de recursos, bueno utilizan la misma pero lo representan de modo diferente.

Cuando veas **un uno (1) como tipo de registro, se corresponde con un registro A (host)**

Hacemos una tabla sencillita para que lo entiendas mejor...

Tipo de registro	Correspondencia	Descripción
1	A (Host)	Se trata de una **máquina** *"normal"*
4	NS	Se trata de un **Servidor DNS**
12	PTR	**Puntero** de resolución inversa
32	SRV	**Localizador de Servicios**

Así podrás interpretarlo mejor. Si te fijas unas líneas más debajo del ejemplo de la pantalla 15, para la entrada de *dns1.sdv.fr* le coloca un *tipo número 4* (*servidor DNS*), la *consulta es inversa (in-addr.arpa*) y en sección coloca *Authority* en lugar de *Answer*.

Las entradas como *Authority* son para los servidores de nombres (DNS SERVERS) que son autoritativos para la zona a que representan, un servidor autoritativo es como "*el que manda*" en ese espacio de direcciones.

Mejor lo digo una vez más, revisa de nuevo el artículo 14 de la serie RAW, protocolo DNS

Cuando veas **entradas con TTL (tiempo de vida) muy grandes**, por ejemplo de 31.536.000 ó parecido, esto es UN AÑO!!, **normalmente son entradas correspondientes a las definiciones del archivo *hosts***.

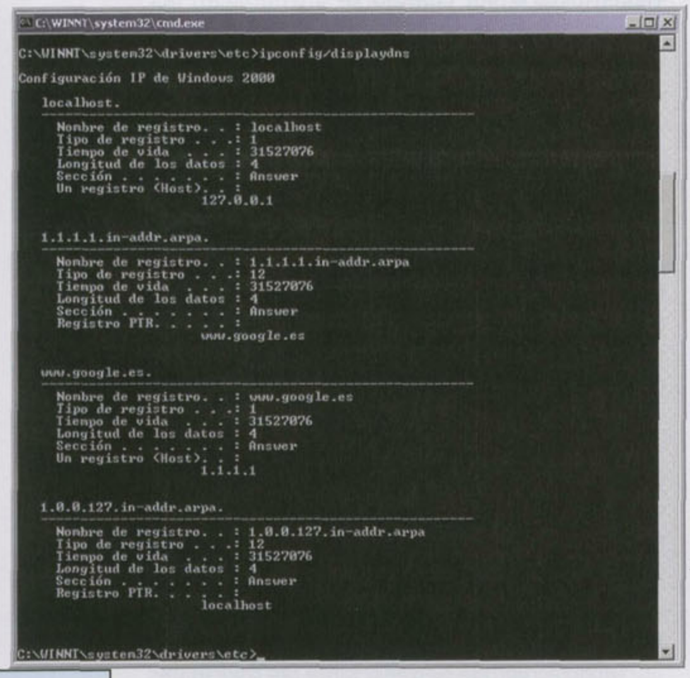
Para eliminar la caché DNS del lado del cliente, se puede usar la orden:

Ipconfig/flushdns

Pantalla 16.- Resultado de la orden ipconfig/flushdns



Y si luego la mostramos de nuevo estará vacía... bueno, no del todo.



Pantalla 17.- Resultado de la orden ipconfig/displaydns

Las resoluciones directas e inversas para la IP 127.0.0.1 (loopback o bucle inverso) siempre existen...

¿Alguien me puede explicar por qué aparece una resolución de tipo A (Host) que relaciona el nombre www.google.es con la IP 1.1.1.1?

Si entendiste bien todo el proceso y las explicaciones que te estoy dando seguro que lo has adivinado...

El administrador de este cliente asignó mediante el archivo *hosts* la IP 1.1.1.1 a www.google.es

El TTL es de "*casi*" un año, algo menos, la diferencia son los segundos que han transcurrido desde que se cargó en caché... pero lo que importa no es eso... **lo que importa es que cuando definimos entradas estáticas mediante el archivo hosts, al vaciar la caché se reasignan los temporizadores y se carga automáticamente en la memoria caché...**

Bueno, creo que ya estamos preparados para lanzar el ataque al DNS (servidor o cliente). Disponemos de abundante información teórica de su funcionamiento y la forma en la que realiza las consultas, planteemos entonces el escenario:

ESCENA DEL CRIMEN

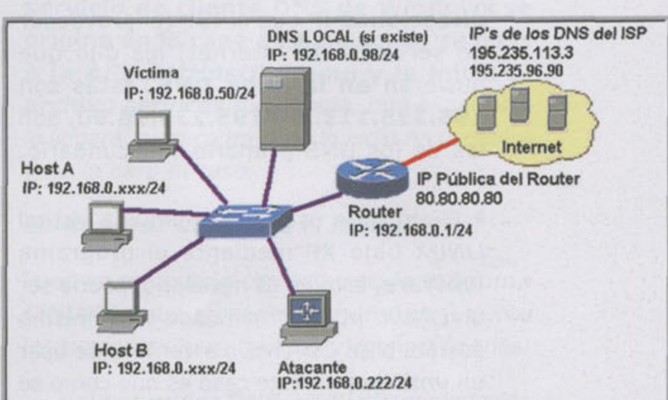


Figura 2.- Escenario de pruebas para el ataque

Antes de nada... no te asustes, **NO NECESITAS TODAS ESAS MÁQUINAS!!!**

Basta con disponer de una víctima y un atacante, una conexión a Internet mediante un router-proxy o conexión compartida y un Switch o Hub para conectarlos a todos.

El hecho de poner más máquinas es para darle un efecto más realista, y de paso, si nos da por envenenar a toda la red o al servidor DNS interno... para el caso es lo mismo, pero vamos por partes, empezaremos por lo "*sencillo*", **el objetivo es este**:

- Supongamos que la **máquina víctima 192.168.0.50** es el PC de trabajo del "*jefe de contabilidad*" y queremos envenenar la caché DNS de ese equipo (sólo de ese). Corre una máquina Windows 2000/XP y se

conecta a su *banco online* para hacer las operaciones bancarias de costumbre en la empresa.

- **El atacante usará la IP 192.168.0.222**, un LiNUX y lo configurará *"ingeniosamente"* con **un servidor BIND** (Servidor DNS), **algunas herramientas de *spoofing*** para envenenamiento ARP y .... **IPTables**.

- **Todas las máquinas de esa red salen a Internet por la puerta de enlace 192.168.0.1** que **es la interface interna del Router**, *he dicho todas... incluido el atacante.*

- **Todas las máquinas** tienen configurado el protocolo TCP/IP con las **dos direcciones** IP que el ISP (el proveedor de servicios de Internet) les dijo que pusieran **en la ficha DNS,** estas son **195.235.113.3 y 195.235.96.90**, son las de los DNS primario y secundario.

- El atacante usará una máquina virtual *LiNUX* bajo XP mediante el programa ***vmWare***, esto no es necesario, podría ser un *LiNUX "normal"* instalado en el mismo portátil o un CD-Live. La ventaja de usar un ***vmWare*** en este caso es que como se ejecuta en una *ventanita* del XP, si *"viene alguien"* basta con minimizar dicha ventana o a las malas, la cerramos y se acabó.

- ***El resto de equipos pueden existir o no***, es indiferente, he puesto un servidor interno DNS porque si en lugar de envenenar la caché del cliente del *"Jefe de Contabilidad"* hacemos lo propio con el Servidor DNS interno, TODOS los equipos de la red se verán afectados... todos, todos, no... el atacante "quedará libre" 😉

Si quieres más información a cerca de cómo instalar una máquina virtual o de cómo crear una Live-CD, mira estos hilos de nuestros foros:

LIVE-CD:
http://www.hackxcrack.com/phpBB2/viewforum.php?f=30
VMWARE:
http://www.hackxcrack.com/phpBB2/viewtopic.php?t=14307

La idea sería disponer de otro equipo por Internet que haga "de *falso servidor web de la banca online*" como esto se nos escapa un poquito del texto, nos conformaremos con envenenar la caché de todos o solamente de la víctima para que cuando acceda a www.microsoft.com (o a la web que nos de la real gana) se dirija a www.redhat.es (que se supone es el espejo falso de la web de Microsoft y que controla plenamente el atacante).

Si conseguimos esto, y suponiendo que lo hacemos creíble, cuando se introduzcan las claves de acceso, contraseñas, etc.. Estaríamos en disposición de saber cuales son, puesto que las registraríamos en nuestro servidor web y no en el verdadero.

IDEAS PARA REALIZAR EL ATAQUE

Veamos lo se nos ocurre....

- **Acceder al archivo *hosts*** de la víctima o víctimas y añadir una entrada estática que asigne la IP de www.redhat.com a www.microsoft.com

  Esto mismo hacía el virus MyDoom en una de sus innumerables "versiones", volvía locos a los infectados con entradas falsas, pero con "salero", incluía direcciones IP incorrectas que pertenecían a las principales casas de antivirus del mercado, así los antivirus no se actualizaban y no podían eliminar el virus una vez que infectó la máquina.... ingenioso, ¿verdad?

- **Manipular el registro de Windows** o la configuración TCP/IP de la víctima para **cambiarle *sus* DNS por los nuestros** y reenviar la petición *"manipulándola"*

- **Cambiar la puerta de enlace de la víctima** que es 192.168.0.1 por la IP 192.168.0.222 que es la del atacante y configurar la máquina *LiNUX* como un router, esto se puede conseguir de varias

formas:

- Cambiando las especificaciones del protocolo TCP/IP en la tarjeta de red o en el registro de Windows
- Manipulando la tabla de rutas y añadiendo una ruta estática
- Enviando mensajes ICMP de tipo REDIRECT desde el atacante haciéndonos pasar por el router dirigidos a la víctima, si acepta los paquetes, cambiará su puerta de enlace dinámicamente por la que decidamos

- Colarle a la víctima ***un troyano con keylogger, esnifer y/o redirector*** de puertos para que nos envíe la información de sus contraseñas y nombre de usuario

......

......

Seguro que se te ocurren muchas más, estas son algunas, y muchas de ellas interesantes, sólo que para que se puedan llevar a la práctica real, necesitaremos escalar privilegios en la *máquina* víctima o disponer de una cuenta de administrador que nos de control total del equipo.

Bueno, para los mensajes ***ICMP Redirect***, para esto no es necesario... pero los *Windows* tienen desactivado "*por defecto*" aceptar ese tipo de paquetes, instalarle *keyloggers*, *esnifers*, troyanos, etc.. Puede no ser viable... es un ataque "*activo*", es un tipo de intrusión en la que necesitaremos saltarnos el antivirus, el firewall si lo hay, instalarle los programas en su ordenador, etc... Lo descartamos por su improbable éxito.

Explicar el funcionamiento de ***ICMP Redirect*** y de otras "*lindezas*" del protocolo *ICMP*, la verdad es que merecería otro artículo "*a parte*" escenificar como manipular la tabla de rutas o los "*usos indebidos*" de ICMP.... otro artículo más para el tintero.... pero para que vayas haciendo boca, mediante mensajes ICMP "manipulados" y enviados a una víctima podemos obligarle a cambiar su puerta de enlace por otra 😉

Quizás, algunos, habréis pensado en enviarle paquetes "*en bruto*" mediante un generador como ***nemesis, excalibur o similares...*** aparentemente puede ser efectivo, *si me hago pasar por el servidor DNS y envío al cliente una respuesta manipulando la asignación de IP, actualizará su caché, ¿no?*

Pues NO, por varios motivos:

1º) Porque aun en el supuesto que construyamos ese paquete "*a medida*" y se lo lancemos al cliente en nombre del servidor DNS, **tendremos que adivinar el puerto dinámico por el que espera recibirlo y el identificador de Transacción** que se usará, como explicamos al principio y eso es dificililllo...

2º) Aunque sepamos esos dos datos o lo probemos "*a lo bestia*" a base de combinaciones, no funcionará, porque **el servicio de cliente DNS de Windows se origina en la capa de Aplicación, se pasa a la *API GetHostByName* y lo inicia el archivo *services.exe***, *o sea, que el paquete le llegará, pero como él no lo pidió no lo quiere y no le hará ni caso.*

¿Entonces? Cual será el método?

Pues yo me quedo con este, ***en la máquina LiNUX*** (la del atacante) tendremos a punto las siguientes herramientas y configuraciones:

- **Un servidor DNS**, un BIND, configurado **especialmente para que resuelva "*mal*"** los nombres de dominio con los que queremos envenenar la red, *vamos un DNS malo, malo, malísimo, que ataca a sus clientes en vez de servirles.*

- ***Dsniff, más concretamente arpspoof***, con esto conseguiremos envenenar ARP, recordemos que estamos en una red conmutada (*con un switch*) y necesitamos "*escuchar*" lo que hace la víctima, *esto fue explicado magistralmente por moebius en la revista número 11, el artículo de Intrusión en redes locales, no obstante lo básico lo explicaré de nuevo*, si la red estuviese conectada por un HUB, no necesitaríamos ni esto.

- ***IPTables***, *otro ejemplo más que esta revista lleva explicando desde hace varios números, no sé si magistralmente, pero*

pormenorizadamente sí que fue, me estoy refiriendo a los artículos de las revistas 21, 22 y 23, sobre todo el 23 que hablaba de NAT, el autor...el mismo que escribe estas líneas. **Necesitaremos IPTables para convertir al LiNUX en un enrutador de paquetes y "algo más"**

Una vez que tengamos todas estas herramientas y servicios configurados al efecto en la máquina atacante, **estaremos en disposición de:**

- **Gracias a *arpspoof*** las comunicaciones de la víctima podrán ser escuchadas por el atacante, ***IPTables*** se encargará de cambiar "*al vuelo*" las peticiones DNS que haga nuestro perdido amigo y las enviará al propio servidor *BIND* que previamente configuramos.

- ***El servidor DNS*** modificará las consultas de la víctima y le entregará las IP's que queramos, para todas sus consultas o para las que decidamos.

- ***IPTables*** volverá a enrutar el paquete convenientemente, se le entregarán las respuestas "*trucadas*" al cliente falseando la dirección del verdadero servidor DNS (*IP-Spoofing*) y nuestro cliente, o todos, o el servidor DNS interno si lo elegimos a él, almacenará en su caché de cliente DNS una entrada positiva... pero con datos falsos, con IP's que apuntan a máquinas que no son las verdaderas.

- Como hemos aprendido, la caché del cliente guardará por un día como mínimo esos datos falsos, hasta podemos desmontar todo el tinglado, es decir, detener el *arpspoof*, *el servidor BIND* o mejor, aun... *apagamos la máquina virtual LiNUX, el engaño seguirá funcionando a menos que reinicie el equipo víctima o vacíe su caché mediante ipconfig/flushdns, cosa bastante improbable por que "el jefe de contabilidad" sabe mucho de lo suyo... pero eso de ipconfig/flushdns le suena a chino... lo de reiniciar está dentro de lo posible... pero para entonces ya habremos conseguido lo que buscábamos y desconectamos el LiNUX :*

Muchos estaréis pensando que bastaría con esnifar las contraseñas y eso... pero recordad que las comunicaciones web iban cifradas en el caso "de la imprenta", o sea, que eso de recoger la cosecha por esa vía, nada de nada...

¿Os gusta la idea? *Pues si os soy sincero, a mi me gusta bastante, es pasiva, no tenemos por qué usar técnicas intrusivas en la víctima como instalarle programas o troyanos, sólo escuchar, enrutar, resolver el nombre, suplantar a los verdaderos servidores DNS, volver a enrutar y a dormir....*

¿Lo hacemos? O.... ¿lo hacéis vosotros?

Ciertamente sería una práctica que todos los que nos habéis seguido estos últimos meses, deberíais ser capaces de realizar por vosotros mismos, exceptuando la configuración del *BIND* que nada hemos dicho hasta hoy, pero lo del *arpspoof* y lo de *IPTables* tendrías que saberlo .... *Ya, nadie piensa en IPTables como "atacante" en lugar de "defensor", qué le vamos hacer, así es la vida... lo explicaré yo, vale.*

PREPARANDO LO QUE NECESITAMOS

Empecemos por preparar las herramientas, le toca a un viejo conocido... ***arpspoof***

Es una herramienta incluida en la suite dsniff, la encontraremos en:

http://monkey.org/~dugsong/dsniff/

Y precisa de numerosas dependencias...

- Berkeley DB en: http://www.sleepycat.com/
- OpenSSL en: http://www.openssl.org/
- Libpcap en: http://www.tcpdump.org/
- Libnet en: http://www.packetfactory.net/Projects/Libnet
- Libnids en: http://www.packetfactory.net/Projects/Libnids

Hay que instalar las dependencias antes de ***dsniff***, hazlo en el orden que te puse, porque unas dependen de otras...

Cuando llegue el momento, tendremos que lanzar esta orden:

arpspoof –i –eth0 –t 192.168.0.50 192.168.0.1

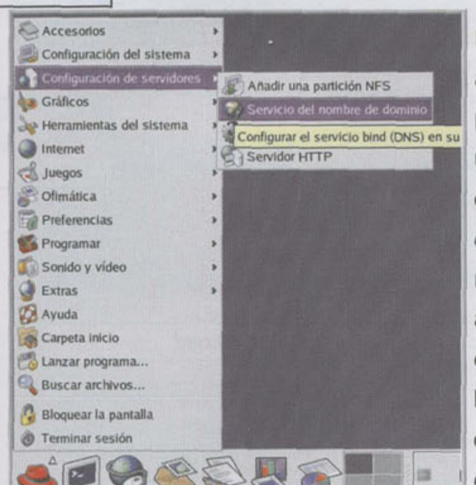
El programa ***arpspoof*** comenzará a enviar paquetes *ARP* haciéndose pasar (***IP-Spoofing***) por la IP 192.168.0.1 (el router) hacia la máquina 192.168.0.50 (la víctima) de tal forma que ésta última actualizará su tabla ARP con la MAC del atacante (192.168.0.222, nuestro *LiNUX*) y TODAS las comunicaciones que realice la víctima con destino al router, las enviará al equipo *LiNUX* en su lugar, con esto hemos conseguido "*ponernos*" en medio... (*MiTM, Man in The Middle*), entremedias del router y de la víctima.

Hará falta algo más, tendremos que hacer llegar esos paquetes al router verdadero y cuando recibamos las respuestas del mismo enviarlos de nuevo a la víctima, *esto es en esencia lo que ya se dijo en el artículo citado de* ***moebius****, sólo que en nuestro caso particular "hay que hacer algo más", ya llegará... cuando le toque el turno a IPTables, un ratito y llegamos a ello.*

Le toca el turno al Servidor BIND

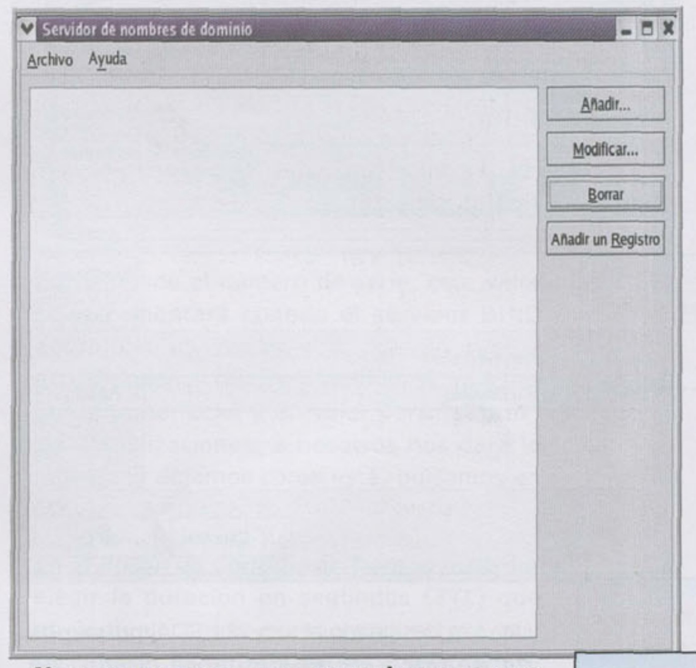
Casi todas las distribuciones de LiNUX incorporan la posibilidad de instalar, configurar y ejecutar un Servidor DNS, un BIND, yo estoy usando RedHat, no creo que tengas dificultades en adaptar el ejemplo al servidor BIND que utilices, también puedes encontrar este tipo de aplicaciones por la web, hasta las hay para Windows....

Podemos comenzar la configuración de un servidor BIND así:



Pantalla 18.- Acceso a la configuración de BIND.

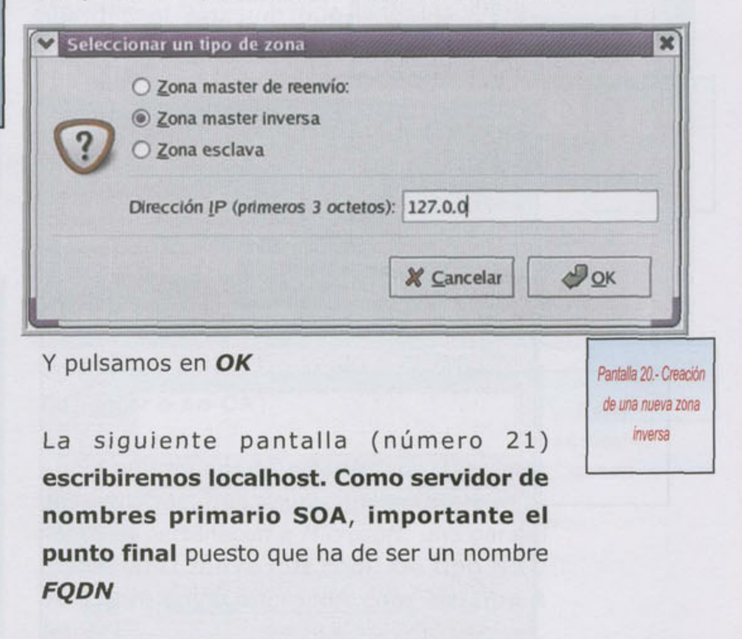
Después te aparecerá una pantalla con una única entrada (si es la primera vez que lo usas), si no es así te aparecerán las que ya se hubiesen definido... el caso es que **ELIMINAREMOS TODAS**, en mi caso usando el ***Botón Borrar*** de la *zona derecha* :P y lo dejamos "*limpio de polvo y paja*" como se muestra a la pantalla 19.



Pantalla 19.- Eliminar TODAS las entradas del servidor BIND

Nos crearemos una zona inversa que apunte a la dirección de loopback de nuestro equipo (127.0.0.1) **y una zona master de reenvío** que será la entrada falsa.

Empezamos por la zona master inversa:



Pantalla 20.- Creación de una nueva zona inversa

Y pulsamos en ***OK***

La siguiente pantalla (número 21) **escribiremos localhost. Como servidor de nombres primario SOA, importante el punto final** puesto que ha de ser un nombre ***FQDN***



Pantalla 21.- Servidor de Nombres FQDN localhost.

Una vez designado el *servidor SOA*, p**ulsamos en Añadir y en el siguiente cuadro de diálogo escribiremos localhosts.** (**Importante otra vez el punto**) en Servido por:

Pantalla 22.- Servido por.....

Pulsaremos en ***OK*** y ***regresamos a la pantalla anterior***, quedará como la que muestra la pantalla número 23:

Pantalla 23.- Configuración de la zona inversa

Ahora ***pulsaremos en el otro Botón de Añadir***, el de la ***Tabla de dirección inversa*** y crearemos esta entrada en el cuadro de diálogo que aparezca (***Nuevo puntero***).

Pantalla 24.- Puntero de localhost.

De nuevo en *OK*, y quedará como figura en la pantalla 25

Pantalla 25.- Configuración completa de la zona inversa

Pulsaremos en ***OK*** y nos tocará crear una ***zona master de reenvío***...

Pantalla 26.- Creación de una zona master de reenvío.

Seleccionamos ***Zona master de reenvío*** y como nombre de dominio el que queramos *"falsear", si recuerdas queremos que cuando el cliente acceda a www.microsoft.com se dirija a www.redhat.es*

La zona de reenvío es la que traducirá los nombres por IP's, mientras que la **zona Inversa en la que traduce IP's en nombres**, por eso elegimos la primera, l**a zona esclava se usa cuando disponemos de varios servidores BIND** y queremos disponer de un respaldo o para actualizaciones, ya te digo que hay mucho que hablar de los servidores DNS... pulsamos en ***OK***.

Aparecerá otro cuadro de diálogo, algunas de las casillas *"ya las cumplimenta"* lo único que tendremos que poner aquí de momento es el ***servidor de nombre primario SOA*** igual que antes, **pero con los datos manipulados...**

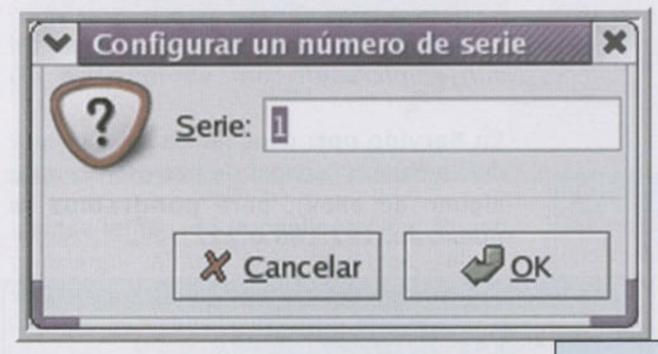
Observa que puse ***ns1.microsoft.com.*** (Termina en un punto) ese será el *FQDN* del Servidor de Nombres.

*Podríamos haber puesto cualquier otra cosa, como **mi.tia.margarita.** Pero mejor "que se aproxime" al verdadero o que parezca "de verdad"*

Pantalla 27.- Servidor SOA para la zona master de reenvío

Nombre para las traducciones IP
Zona master
Nombre: microsoft.com
Nombre de ficheros: microsoft.com.zone
Contacto: root@localhost
Servidor de nombre primario (SOA): ns1.microsoft.com.
Número de serie: 1 Configurar...
Configuración del tiempo...
Registros
microsoft.com
Añadir... Modificar... Borrar
Cancelar OK

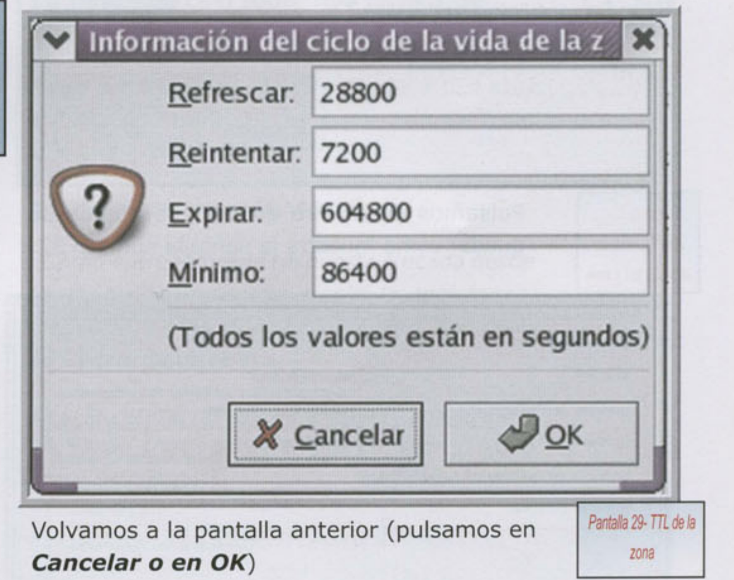
Si pulsamos en ***Configurar***, podremos establecer un valor numérico



Pantalla 28.- Número de serie

Corresponde al número de serie, este valor se incrementará cuando el servidor BIND actualice las zonas y o cuando reciba actualizaciones de otros servidores, muchos utilizan una fecha y un valor para "**seguir**" las actualizaciones, a nosotros nos dará lo mismo, lo dejamos como está, pulsamos en ***OK***

En el Botón de Configurar Tiempo podemos elegir la duración en segundos (*TTL*) que enviará al cliente como respuesta a sus solicitudes, no modificaremos nada, pero te pongo la pantalla para que lo veas:



Pantalla 29- TTL de la zona

Volvamos a la pantalla anterior (pulsamos en ***Cancelar o en OK***)

Ahora pulsaremos en ***Añadir***, nos crearemos dos registros, que apunten a las IP's que se suponen pertenecen a *Microsoft*, una para el propio ***dns1.microsoft.com.* De tipo NS** y **otra *para* *www.microsoft.com* *del tipo A (Host)***

En el *desplegable* de ***Añadir recurso registrado*** elegimos ***Servidor de nombres***

En nombre de dominio ponemos ***ns1*** (*el sufijo microsoft.com ya lo pone él*)

En Servido por: deberíamos colocar la IP del verdadero servidor de microsoft.com (o alguno de ellos), pero **pondremos la nuestra... 192.168.0.222**

Pantalla 30.- Datos para el registro de ns1.microsoft.com

Añadir un registro
Añadir el recurso registrado: Servidor de nombres
Nombre de dominio: ns1 .microsoft.com
Servido por: 192.168.0.222
Cancelar OK

Pulsamos en OK y ***repetimos el mismo proceso pero para el recurso www de tipo hosts***



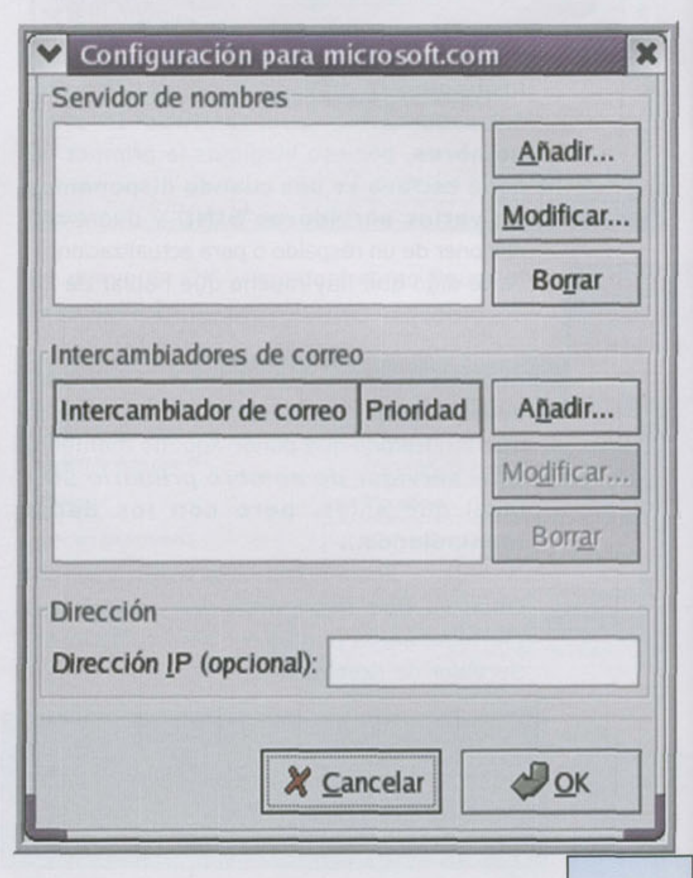
Pantalla 31.- datos para el registro de tipo www

Pulsamos ***de nuevo en OK*** y la nos ha de quedar como muestra la pantalla número 32:

Pantalla 32.- Resultados de las entradas ns1 y www

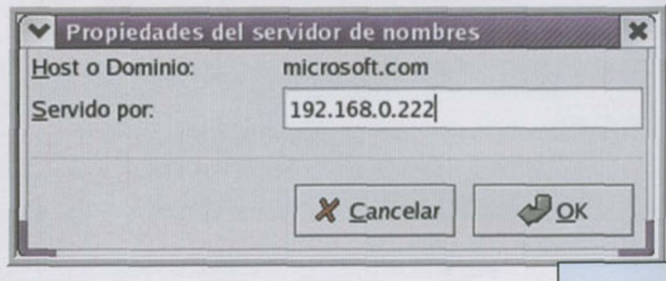
Nombre para las traducciones IP
Zona master
Nombre: microsoft.com
Nombre de ficheros: microsoft.com.zone
Contacto: root@localhost
Servidor de nombre primario (SOA):
Número de serie: 1 Configurar...
Configuración del tiempo...
Registros
microsoft.com
ns1 servido por 192.168.0.222
www servido por 66.187.229.25
Añadir... Modificar... Borrar
Cancelar OK

Resaltamos en la zona de Registros el nombre microsoft.com (como muestra la pantalla número 32) y pulsamos en ***Modificar***, verás esto:



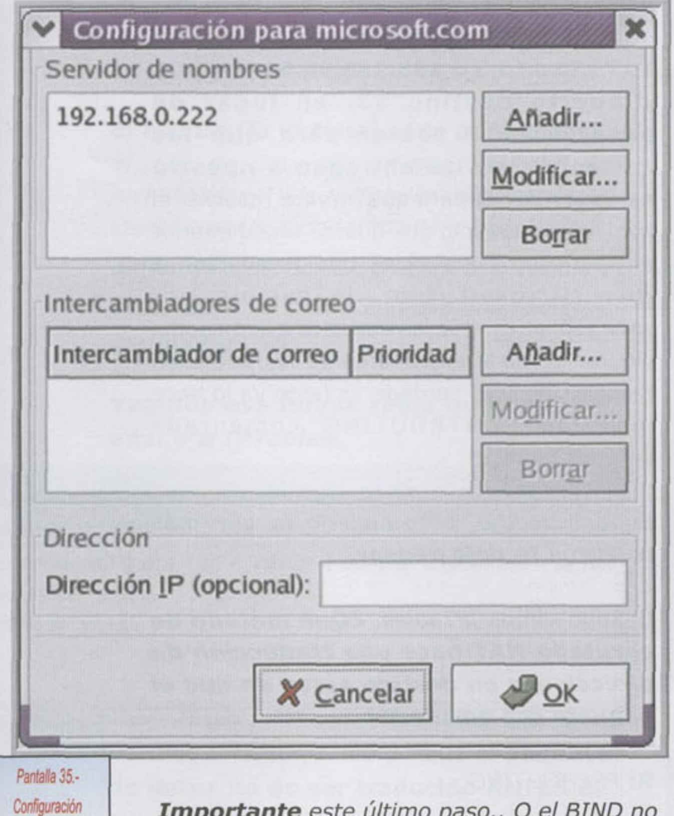
Pantalla 33.- Configuración para microsoft.com

Pulsamos en el ***Botón superior de Añadir en la zona de servidor de nombres***, la otra sería para incluir registros *MX* para el servidor o servidores de correo.



Pantalla 34.- IP para buscar el dominio microsoft.com

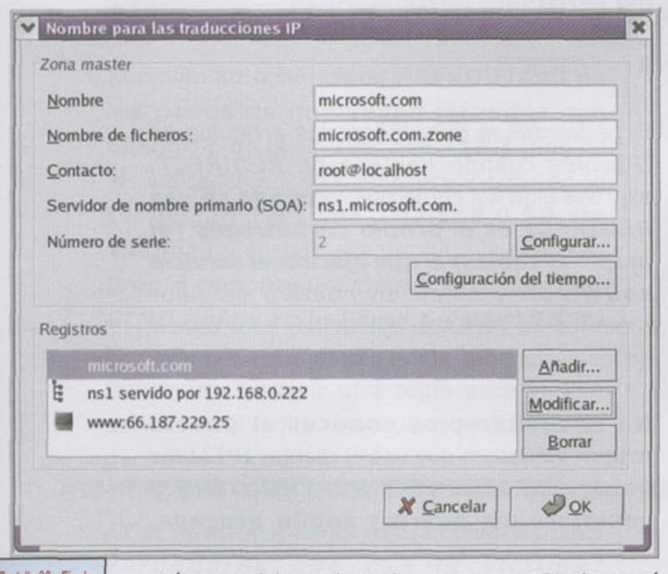
Y cuando pulsamos en ***OK*** (*y van tres mil veces ya* ☺) aparecerá una pantalla como la que muestra la número 35:



Pantalla 35.- Configuración finalizada de microsoft.com

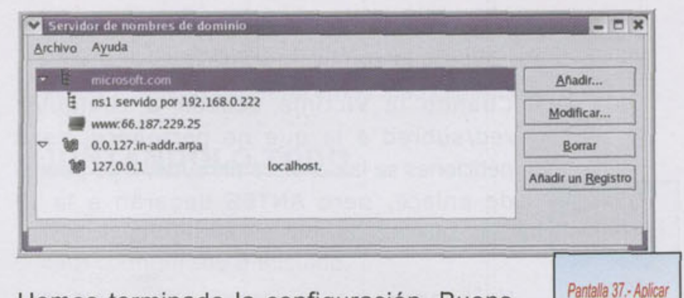
Importante *este último paso.. O el BIND no hará lo que esperamos*

Te lo imaginas, ¿no? Eso, otra vez OK (ya terminamos)



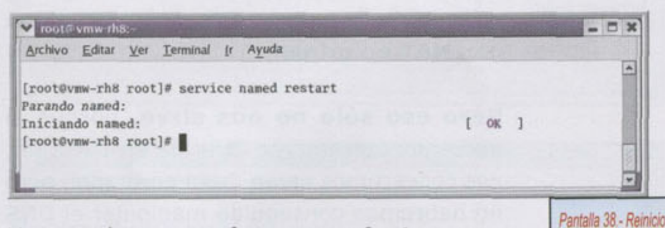
Pantalla 36.- Fin de proceso para la creación de la zona master de reenvío

Sólo nos falta pulsar ***de nuevo en OK*** (*esta sí que será la última vez*) y en la siguiente pantalla (número 37) en el ***menú de archivo, Aplicar.***



Pantalla 37.- Aplicar los cambios y fin de configuración

Hemos terminado la configuración. Bueno, podríamos añadir más registros, falsos, verdaderos, inexistentes o como quieras, pero esto es la *Configuración*, ***hay que iniciar el servicio (service named restart o en muchas otras distribuciones LiNUX, /etc/init.d/named restart***



Pantalla 38.- Reinicio del servidor named.

Y ya sabes, ***service named stop*** *para detenerlo, o reload para recargar las zonas, probe, status, start....*

Lo tenemos casi todo, nos falta algo...

¿Cómo podemos explicar a la víctima que tiene configurados unos DNS 195.235.113.3 y 195.235.96.90 que en lugar de usar esos utilice el nuestro?

Y aunque consigamos eso...

¿Cómo entregamos la respuesta trucada desde la máquina atacante hacia la victima si ***desconocemos el puerto origen que usa, el id-transaction****?*

¿Y si la víctima no accede a www.microsoft.com ? ¿Serán todas sus peticiones redirigidas a www.redhat.es ?

¿Lo sabes ya? **ENHORABUENA!!!**

Contestemos...para los que no hayáis dado con la respuesta:

Al lanzar *nuestro* ***arpspoof -i eth0 -t 192.168.0.50 192.168.0.1*** ya hemos dicho que las comunicaciones cuyo origen sea la 192.168.0.50 (la victima) y con destino

192.168.0.1 (el router) pasarán por nosotros, por la IP 192.168.0.222

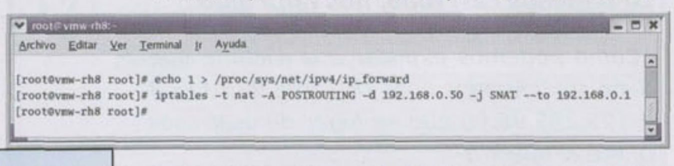
Cuando la víctima acceda a cualquier red/subred a la que no pertenece, esas peticiones se las enviará al router, a su puerta de enlace, pero ANTES llegarán a la IP 192.168.0.222.

Ya tenemos la primera baza ganada, ahora tenemos que ser capaces de reenviar al router las peticiones que pasan por nosotros y cuando regresen hacerlas llegar de nuevo a la víctima.

Eso nos lo solucionó ***moebius*** en su artículo...la respuesta es simple: **IPTables con NAT en origen**,

Pero eso sólo no nos sirve, porque si ponemos únicamente la regla en cuestión, nos convertimos en un mero enrutador, pero no habremos conseguido manipular el DNS ni cambiar la resolución de nombres...

Si atendemos a las explicaciones del artículo *famoso*, tendríamos que hacer esto:



Pantalla 39.- Habilitar el reenvío y traducción NAT en origen.

La primera línea habilita el reenvío de paquetes por la interface de nuestro LiNUX, vale con esto la máquina del atacante *"sabe"* que ha de encaminar paquetes...

Voy a citar las palabras textuales del artículo de la revista número 11 (con tu permiso moebius).

"En el momento que el paquete esté listo para salir (POSTROUTING) cambiará la IP origen (-j SNAT) y usará la de la máquina suplantada (--to 192.168.0.1)"

O sea, que la víctima recibirá los paquetes como si realmente viniesen del router aunque realmente se la está enviando el atacante.

¿Qué nos falta?

Pues **explicarle a IPTables que si los paquetes van con destino a las IP's 195.235.113.3 ó 195.235.96.90 por UDP y puerto destino 53, en lugar de pasárselas al router para que las encamine, se las entregue a nuestro servidor BIND** para que intente resolver el nombre de dominio que guarda en su interior el paquete UDP y si es una resolución a www.microsoft.com lo cambie por www.redhat.es y se las devuelva a la máquina víctima como si efectivamente las hubiesen resuelto los DNS verdaderos (esto ya lo hace la regla POSTROUTING comentada anteriormente).

Es fácil decirlo, pero hacerlo es aún más sencillo... ***te dejo pensar....***

Cuando vimos *IPTables*, ***¿Qué método de enrutado NAT hace una traducción de direcciones en destino antes de que el paquete sea enrutado?***

R: *PREROUTING*

Y.... ***¿qué reglas de IPTables entregan los paquetes al propio host cortafuegos (en nuestro caso el atacante) para que sea éste quien ejecute el servicio en lugar del verdadero destinatario?***

R: *REDIRECT*

Si recuerdas lo que dije en los artículos de IPTables cuando hablaba de *REDIRECT*, expresé algo así como que, ***cuando se usa REDIRECT es el propio cortafuegos (el propio IPTables) el que ejecuta el servicio solicitado*** y como en nuestro servidor tenemos un BIND en ejecución, será éste y no los designados por el cliente quien lo haga.

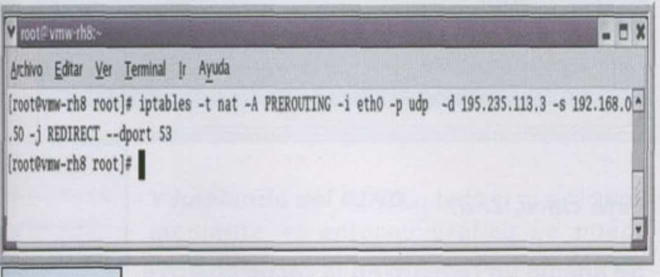
No necesitaremos conocer el puerto origen verdadero que usó el cliente, ***IPTables hará PREROUTING Y POSTROUTING y cambiará los puertos según proceda.***

También nos despreocuparemos del ID-Transaction del protocolo DNS, **será nuestro servidor BIND quien lo recoja y responda**

usando el mismo identificador y como dispone de un registro que apunta a una IP "*engañosa*" le servirá esa y no la verdadera.

El cliente (la víctima) ni se dará cuenta*, el pide un dato por un puerto y con un identificador X y le responden con ese mismo identificador (gracias al BIND) y por el puerto dinámico que abrió (gracias a IPTables), actualizará su caché con esa asignación y* ***ENVENENADO.***

Veamos esa nueva regla que hay que añadir a *IPTables*:



Pantalla 40.- NAT en destino con PREROUTING

Usamos ***PREROUTING*** porque **el paquete de datos ha de ser traducido ANTES de ser enrutado.**

Como sólo nos interesa hacer REDIRECT aquello que sea UDP y puerto 53, utilizamos:

-p udp (protocolo udp)

--dport 53 (aquello que vaya dirigido al puerto 53)

Además, **solo haremos PREROUTING de los paquetes que vayan dirigidos con destino (-d) la IP 195.235.113.3** (uno de los verdaderos DNS), **siempre y cuando la IP origen (-s) sea 192.168.0.50 (la víctima)** y si todo eso se cumple, ***será el propio cortafuegos mediante el servidor BIND quien se lo coma (-j REDIRECT)***

Deberíamos incluir otra regla para el otro servidor DNS que usa nuestra víctima, es decir:

iptables -t nat -A PREROUTING -i eth0 -p udp - d 195.235.96.90 -s 192.168.0.50 - j \ REDIRECT --dport 53

¿Y si desconocemos los servidores DNS verdaderos que usa la víctima?

Venga... no me digas que no lo sabes.... ni lo comento, sencillamente te incluyo la regla:

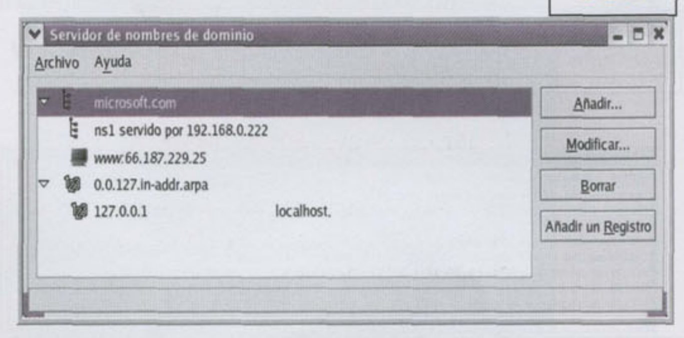
iptables –t nat –A PREROUTING –i eth0 –p udp –s 192.168.0.50 – j REDIRECT –dport 53

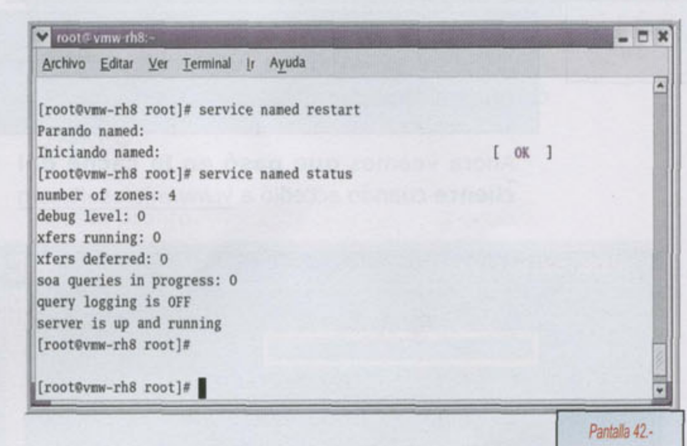
Y funcionará.... Vamos a juntarlo todo, nos creamos un script y lanzamos el ataque "*de verdad*", luego observaremos los resultados.

JUNTÁNDOLO TODO

Nos aseguramos de que el servicio named está configurado e iniciado.

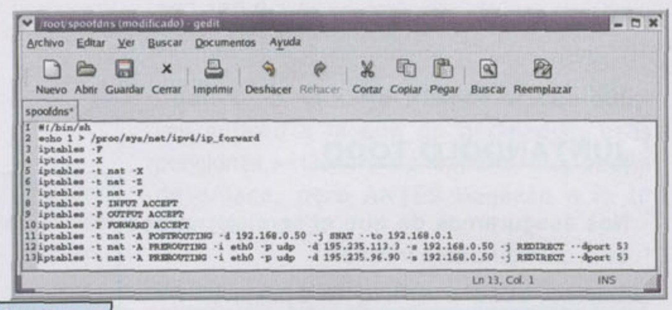
Pantalla 41.- Comprobación de la configuración del servidor DNS





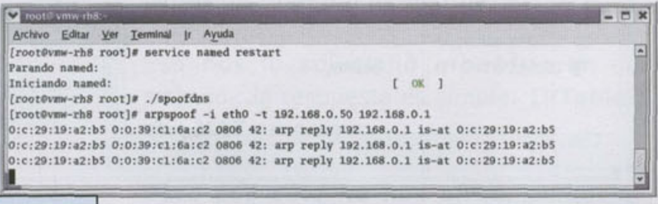
Pantalla 42.- Reiniciamos y comprobamos el funcionamiento del servidor DNS

Vamos a crearnos un *script* con las definiciones de las tablas, cadenas y filtros para *IPTables* y lo llamamos ***spoofdns*** (*por qué no* ☺), observa que empezamos eliminando cualquier otra anterior, así nos aseguramos que no hay otras anteriores que entorpezcan a las nuevas.



Pantalla 43.- Creación del script ./spoofdns

Lanzamos nuestro *script* ***./spoofdns y el arpspoof****....* lo dejamos correr...



Pantalla 44.- Todo junto....

Recuerda*... no detengas la ejecución de arpspoof, al menos no hasta que la víctima caiga en nuestras "redes"*

Ahora veamos **que pasó en la caché del cliente** cuando accedió a www.microsoft.com

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Victor>ping www.microsoft.com

Haciendo ping a www.microsoft.com [66.187.229.25] con 32 bytes de datos:

Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.

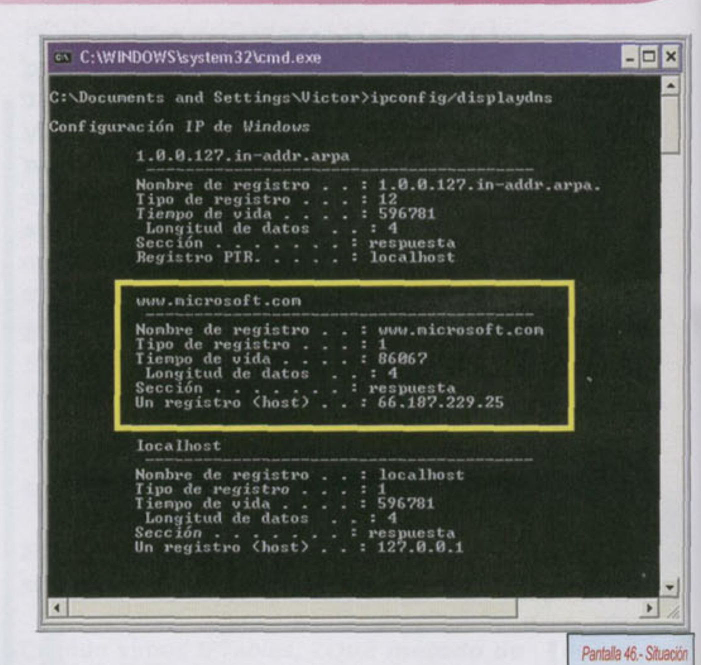
Estadísticas de ping para 66.187.229.25:
    Paquetes: enviados = 4, recibidos = 0, perdidos = 4
    (100% perdidos),

C:\Documents and Settings\Victor>_
```

Pantalla 45.- El cliente envía un ping a www.microsoft.com

Que no responda al ping no es porque hayamos hecho algo mal, hay muchos servidores de Internet que están así configurados, no es un fallo en nuestra configuración... ***lo que sí es relevante es que la IP que resuelve es 66.187.229.25 que NO ES LA DE MICROSOFT****, es el registro "falso" que nos creamos en nuestro servidor* ***BIND, IPTables + arpspoof hicieron su trabajo convenientemente***

Ahora veamos cómo está la caché:

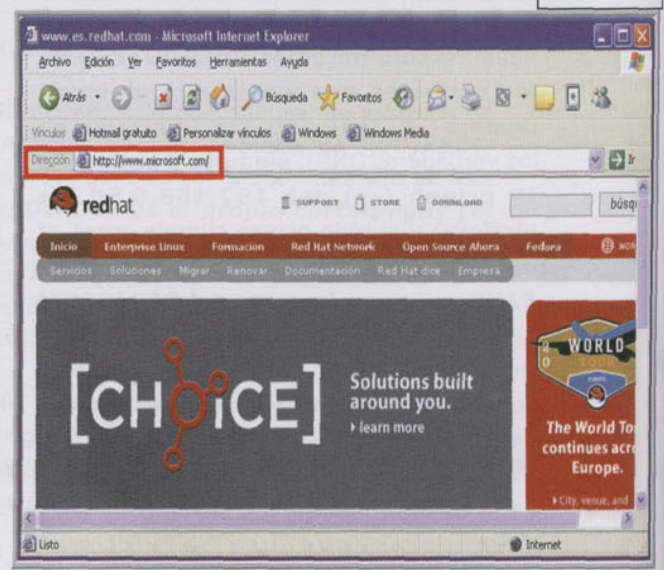


Pantalla 46.- Situación de la caché DNS del lado del cliente

Está claro, ¿no?

Se añadió un registro en la caché positiva del cliente que apunta a la IP de www.redhat.es como si se tratase de www.microsoft.com

Si en lugar de un ping, se hubiese navegado hacia www.microsoft.com, esto es lo que le aparecería al cliente:



Pantalla 47.- Acceso a www.microsoft.com

Observa la barra de dirección (rectangulo rojo).... sin comentarios....

***Los navegadores** suelen mantener también otra caché... es posible que aparezca la verdadera web de Microsoft pero cuando se pulse en cualquier link de esa página dará un error, entonces lo más habitual es que se pulse en actualizar o F5 ó CTRL.+F5 si se trata de un Internet Explorer, entonces aparecerá la web de RedHat.*

Por eso decía que si hacemos un *phising* de la web a falsear y nos montamos una idéntica con los mismos enlaces, etc... La víctima NO se dará ni cuenta, existen herramientas que pueden "*descargarse*" una web enterita, desde aplicaciones de desarrollo como *Dreamweaver*, herramientas libres o "*semi libres*" como ***teleport***, etc... Con ellas podemos llevarnos una web al disco local y luego colocarla en el falso servidor al que apunta nuestro *BIND*

Y hablando del BIND... todo lo que hicimos mediante el entorno gráfico se puede configurar de un modo más rápido y sencillo en modo texto, échale un vistazo a los archivos que se crearon...

Pantalla 48.- Archivos de configuración de zonas del servidor named

```
root@vmw-rh8:/var/named
Archivo  Editar  Ver  Terminal  Ir  Ayuda
[root@vmw-rh8 root]# cd /etc
[root@vmw-rh8 etc]# ls named*
named.conf  named.conf~  named.custom
[root@vmw-rh8 etc]# cd /var/named
[root@vmw-rh8 named]# ls
0.0.127.in-addr.arpa.zone     forward.example.com.zone  named.local
0.0.127.in-addr.arpa.zone~    localhost.zone            terra.es.zone
0.28.172.in-addr.arpa.zone    microsoft.com.zone        www.terra.es.zone
153.194.213.in-addr.arpa.zone named.ca
[root@vmw-rh8 named]#
```

Son archivos que se pueden editar con *gedit*, *vi*, *pico* o el que más te guste.

***Un último detalle**, si tras lanzar el ataque y una vez que haya caído la víctima paramos TODO, hasta si apagamos el LiNUX, SEGUIRÁ FUNCIONANDO para clientes DNS Windows durante un día entero... recuerda lo del TTL en caché, será así hasta que transcurra ese tiempo o reinicie el equipo Windows, vacíe la caché DNS con /flushdns o deshabilite y vuelva activar la tarjeta de red...*

Si la víctima fuese un cliente DNS LiNUX, tendríamos que continuar con arpspoof todo el tiempo, puesto que los clientes LiNUX no usan caché DNS.

Bueno, pues esto se acabó... ha sido duro... lo sé... pero creo que mereció la pena (eso creo)

Hasta pronto

Vic_Thor.

CURSO DE TCP IP

LA CAPA IP 3ª PARTE : FRAGMENTACION DE DATAGRAMAS

Este mes hay MUCHA miga:
- Nos meteremos de lleno en los algoritmos de fragmentación y reensambado de paquetes
- Trataremos los problemas de seguridad derivados de la fragmentación de datagramas
- DoS / Exploits Jolt, Jolt2 y TEARDROP / Floog Y Nuke
- Saltaremos firewalls e IDSs



INTRODUCCIÓN

Cada vez me voy sintiendo más a gusto escribiendo para esta revista, ya que hemos avanzado juntos ya muchísimos pasos durante todos estos meses de curso TCP/IP y también, por supuesto, con la serie RAW.

Esto hace que cada vez pueda hablar con más soltura, sin pensar en todo momento si los lectores conocerán tal cosa a la que hago referencia. Mis referencias son cada vez más a artículos ya escritos, más que a cosas totalmente desconocidas para vosotros. Esto me da muchísima más libertad para centrarme en temas más específicos, y este artículo es buena prueba de ello.

Difícilmente habría podido escribir este artículo sin asumir que partís ya de una base bastante sólida.

Quizá os parezca que abuso un poco de vosotros, ya que para poder seguir este artículo enterándose de todo no sólo es imprescindible haber seguido mi **curso de TCP/IP**, si no que también asumo que habéis seguido el resto de la revista.

Para seguir este artículo es necesario que hayáis seguido el imprescindible **curso de Linux** de la revista (ya que centro en Linux mis ejemplos), así como el magnífico **curso de diseño de firewalls** (basado en **iptables**), y otros artículos, como los que hablaban del uso de **exploits**.

Todos estos ingredientes hacen que el nivel de este artículo sea ya bastante alto y, por tanto, también bastante interesante, en mi opinión.

A pesar de eso, me he reservado algunos de los detalles más específicos para una próxima entrega, en la que quiero, entre otras cosas, hacer un compendio de las técnicas de hacking que utilizan ip-spoofing, para ir explicándolas con ejemplos concretos.

A pesar de lo específico del tema del artículo, me ha faltado espacio para explicar con mucho más detalle muchas cosas, pero siempre debéis considerar esta revista como un punto de partida para vuestro estudio personal, con el gran maestro Google, y con vuestra propia experimentación, que es absolutamente fundamental.

Una vez más he de insistir en que los que busquen un tutorial sobre cómo destruir una máquina escribiendo un comando o ejecutando una aplicación llena de ventanitas y dibujitos, que no pierdan el tiempo leyendo mi artículo.

Lo que aquí se explica no sólo en muchos casos ni siquiera tendrá aplicación práctica, si no que además todo el enfoque del artículo no es desde un punto de vista práctico, si no orientado a aquellos que, como yo, disfrutan conociendo el funcionamiento de las cosas, incluso aunque sean cosas que ya han quedado obsoletas.

1. ¿Es deseable la fragmentación de datagramas?

La respuesta a la pregunta que planteo aquí es sencilla: **no**, no es deseable la fragmentación.

La fragmentación da la oportunidad a un "hacker" de llevar a cabo un gran número de ataques, como ataques **DoS**, o técnicas para **saltarse la protección de un firewall** o un IDS.

Para comprender todo esto, voy a dividir el artículo en tres partes:

- En primer lugar, especificaré detalladamente los algoritmos utilizados para fragmentar y reensamblar datagramas.
- En segundo lugar, presentaré algunas técnicas utilizadas para aprovechar el mecanismo de fragmentación en un ataque.
- Por último, explicaré como configurar un sistema para evitar la fragmentación y, por tanto, cerrar la posibilidad de sufrir este tipo de ataques.

2. Algoritmos de fragmentación y reensamblado

En el **RFC 791** se explica en detalle un algoritmo para la fragmentación de datagramas, y otro para su reensamblado. Si habéis ojeado el RFC, estoy seguro de que la mayoría de vosotros habréis pasado ese punto al encontrar un montón de letras raras que no os interesaban lo más mínimo.



Para los nuevos...

Para los nuevos lectores (y perdonen los curtidos veteranos por repetir una vez más esta nota en cada número de la revista) informarles que en la Web www.rfc-es.org tienen a su disposición el RFC 791 en perfecto castellano.

Una vez más agradecemos a todos los colaboradores de www.rfc-es.org la aportación desinteresada de las traducciones de los RFCs. Traducir un RFC del Ingles al castellano es una tarea larga y compleja, pero hacerlo con la pulcritud que ellos lo hacen es merecedor de nuestros mejores halagos.

Si algún lector desea colaborar con ellos en su gran proyecto, seguro que sois recibidos con los brazos abiertos.

Pero, igual que quedamos en que íbamos a tratar de comprender todo el RFC, también tenéis que recordar que lo que nos diferencia a nosotros del resto de los usuarios de ordenadores son precisamente estas cosas.

Un "usuario" considera el ordenador una **herramienta.** No tiene ningún interés en cómo funcione por dentro, y cuanto más masticado le dé todo el ordenador, mucho mejor para él.

En cambio, para nosotros... esta bien, digamos para un "hacker", el ordenador no es sólo una herramienta para conseguir algo que nos interesa, si no que el propio ordenador nos resulta **interesante**. Por tanto, lo que nos diferencia es que mientras que los usuarios comunes intentan abstraerse lo máximo posible del funcionamiento interno de la máquina, nosotros al contrario buscamos conocer cada vez con más detalle este funcionamiento.

Así que, una vez explicado a grandes rasgos el mecanismo de fragmentación de datagramas, nuestras mentes inquietas no se conformarán con eso, y buscarán ávidamente cada vez más detalle. Este detalle es el que nos da el RFC, y más abajo no podemos llegar, ya que especifica detalladamente un algoritmo que puede ser implementado directamente por cualquier programador.
Por tanto, vamos a tratar de comprender estos algoritmos. 😉

2.1. Algoritmo de fragmentación de datagramas del RFC 791.

Pero antes de nada... ¿hay alguien aquí que no sepa lo que es un **algoritmo**?

Si es así, no os asustéis, que no tiene nada que ver con las matemáticas (uno de los

errores más comunes de la gente no acostumbrada a tratar con estos términos es confundir la palabra algoritmo con logaritmo).

Un algoritmo es simplemente la explicación detallada de una **forma de resolver un problema.**

Por ejemplo, vamos a ver un algoritmo sencillo que tenemos todos implementados en nuestro cerebro, y lo usamos a diario: **el algoritmo para cruzar una calle**.

Un **algoritmo** suele venir detallado por una serie de pasos secuenciales. Se empieza en un primer paso, en el que se llevan a cabo unas acciones. Cuando se termina este paso se sigue en el paso siguiente, y así sucesivamente.

En algunos pasos puede haber saltos que te lleven atrás a un paso que ya hicimos anteriormente, o bien adelante, saltándonos así varios pasos que no es necesario que llevemos a cabo por los motivos que sean.

En cada paso puede haber acciones a realizar, o bien preguntas a responder acerca del estado del problema. Las respuestas a estas preguntas nos llevarán por un camino u otro dentro del algoritmo, moviéndonos de un paso a otro no siempre de forma secuencial, hasta que lleguemos a algún paso que termine el algoritmo (un paso de **FIN**).

Veamos, por tanto, el algoritmo que utilizamos nosotros para cruzar la calle:

Paso 1: Buscamos un paso de peatones y nos acercamos a él, situándonos en el borde de la acera.

Paso 2: Miramos a la izquierda.

Paso 3:
--> Si viene algún coche por la izquierda, entonces volvemos al **Paso 2**.
-->Si no viene ningún coche por la izquierda, entonces continuamos por el **Paso 4**.

Paso 4: Miramos a la derecha.

Paso 5 -
--> Si viene algún coche por la derecha, entonces pasamos al **Paso 7**.
--> Si no viene ningún coche por la derecha, entonces continuamos por el **Paso 6**.

Paso 6: Podemos cruzar la calle. **FIN**.

Paso 7: Miramos a la derecha. La diferencia de este paso con el **4**, es que como aquí permaneceremos un tiempo mirando a la derecha, durante este tiempo podría volver a haber coches en el lado izquierdo, por lo que antes de cruzar tendremos que volver a comprobar este lado, tal y como veremos ahora mismo.

Paso 8:
--> Si no viene ningún coche por la derecha, entonces volvemos al **Paso 2**.
--> Si viene algún coche por la derecha, entonces volvemos al **Paso 7**.

No creo que haya ninguna duda sobre el funcionamiento de este algoritmo, ya que todos lo utilizamos casi inconscientemente. Un programador se tiene que encontrar constantemente con este problema de racionalizar y detallar en un algoritmo muchas cosas que hacemos de forma inconsciente sin reparar en la complejidad de lo que estamos haciendo.

Por supuesto, este algoritmo para cruzar una calle es muy simple, y el que utilizamos nosotros realmente es mucho más complejo. Nuestro cerebro tiene en cuenta mil factores más: que la calle sea de doble o único sentido, que haya semáforos, que los coches estén atascados, etc, etc.

Ahora que ya comprendemos el mecanismo básico de un algoritmo, podemos enfrentarnos al **algoritmo de fragmentación** planteado en el RFC.

Para ello, en primer lugar tenemos que dar nombres a una serie de **variables** que se van a utilizar en el algoritmo.

TL - Longitud total del datagrama (antes de ser fragmentado). *[Total Length]*

MTU – Más adelante hablaremos mucho más sobre la MTU. La MTU es el tamaño máximo que puede tener un datagrama según la tecnología de la red que va a tener que atravesar en su próximo paso. *[Maximum Transmission Unit]*

DF – Flag de la cabecera IP que obliga a que un datagrama no pueda ser fragmentado. *[Dont Fragment]*

IHL – Campo de la cabecera IP que especifica el tamaño de esta cabecera. A lo largo del algoritmo, se referirá en unos momentos al IHL del datagrama original, o al IHL del fragmento, tal y como veremos. *[Internet Header Length]*

OIHL – Variable de uso auxiliar, que nos permite recordar el tamaño de la cabecera IP que tenía el datagrama antes de ser fragmentado. *[Old Internet Header Length]*

OTL – Variable de uso auxiliar, que nos permite recordar el tamaño total del datagrama antes de ser fragmentado. *[Old Total Length]*

FO – Campo de la cabecera *IP Fragment Offset*, que nos permite saber qué posición ocupa un fragmento dentro del datagrama original. *[Fragment Offset]*

OFO – Variable de uso auxiliar, que nos permite recordar el *Fragment Offset* del datagrama antes de ser fragmentado. *[Old Fragment Offset]*

MF – Flag de la cabecera IP que indica que este fragmento no es el último, y aún han de venir más fragmentos del mismo datagrama. *[More Fragments]*

OMF – Variable de uso auxiliar, que nos permite recordar el flag MF del datagrama antes de ser fragmentado. *[Old More Fragments]*

NFB – Número de bloques de datos de los que se compone un fragmento. *[Number of Fragment Blocks]*

Vamos a ver ahora el algoritmo detallado. Os recomiendo que según lo voy explicando vayáis siguiendo cada paso del algoritmo tal y como viene en el RFC (bastante más ofuscado de lo que lo explicaré yo aquí), para que os acostumbréis a la notación utilizada en los RFCs.

El algoritmo del RFC muestra sólo el caso más simple, de un datagrama que se divida en sólo dos fragmentos.

Empecemos viendo el primer paso, que llamaremos **Paso 0** para que se correspondan el resto de pasos con los del RFC:

<u>Paso 0</u> – Si *TL* es menor o igual que la *MTU*, entonces el datagrama cabrá *por* la red y *no* habrá que fragmentarlo. **FIN.**

En el caso contrario (*TL* > *MTU*), entonces tenemos que comprobar el flag *DF*. Si el flag *DF* está activo, no podemos fragmentar, por lo que tenemos que descartar este datagrama, y enviar un ICMP para notificar el problema. **FIN.**

Si el flag *DF* no está activo, entonces podemos continuar por el **Paso 1**.

Este paso, tal y como está escrito, es como solemos hablar nosotros informalmente. Pero esta no es una buena forma de detallar un algoritmo, ya que se presta a muchas ambigüedades.

Una forma mucho más correcta es utilizar ciertas estructuras bien conocidas para detallar la secuencia que siguen los pasos del algoritmo. La estructura más simple para detallar el flujo del algoritmo es el **IF THEN, ELSE THEN**. Lo traduciremos por **SI ENTONCES, SI NO ENTONCES**. Vamos a ver este mismo paso, pero detallado con esta estructura:

Paso 0 –

SI *TL* es menor o igual que la *MTU* **ENTONCES**
El datagrama cabrá por la red y no habrá que fragmentarlo.
FIN.
SI NO, ENTONCES
Comprobar el flag *DF*.
SI el flag *DF* está activo **ENTONCES**
No podemos fragmentar. Descartar el datagrama. Enviar un ICMP para notificar el problema.
FIN.
SI NO, ENTONCES
Continuar por el **Paso 1**.

¿Verdad que esto es mucho más claro una vez que nos acostumbramos a leerlo así?

Además, tal y como está escrito esto, se puede escribir de forma casi directa en cualquier lenguaje de programación, por lo que facilitamos mucho el trabajo a los programadores que tengan que implementar este algoritmo en una máquina.

Veamos ya el algoritmo completo:

Paso 0 –

SI *TL* es menor o igual que la *MTU* **ENTONCES**
El datagrama cabrá por la red y no habrá que fragmentarlo.
FIN.
SI NO ENTONCES
Comprobar el flag *DF*.
SI el flag *DF* está activo **ENTONCES**
No podemos fragmentar. Descartar el datagrama. Enviar un ICMP para notificar el problema.
FIN.
SI NO ENTONCES
Continuar por el **Paso 1**.

Paso 1 –

Copia toda la cabecera IP al primer fragmento que vamos a crear.

Paso 2 –

Damos valor a las variables auxiliares, ya que necesitamos seguir conociendo los datos del datagrama original, pero tenemos que modificar las variables *TL, IHL, FO*, y *MF* para nuestro nuevo fragmento.

OIHL = IHL.
OTL = TL.
OFO = FO
OMF = MF

Paso 3 –

NFB = (MTU - IHL x 4) / 8.
Calculamos el tamaño de los datos del fragmento. El tamaño máximo para todo el datagrama está determinado por la *MTU*. Como, además de los datos, hay que incluir la cabecera, el tamaño de los datos estará en función de la *MTU* y del tamaño de la cabecera (*IHL*).

Paso 4-

Añadir al fragmento *NFB x 8* bytes de datos.

Ahora ya no sólo tenemos la cabecera del fragmento, si no también los datos, que son solo una porción de todos los datos que había en el datagrama original.

Paso 5-

Ahora que ya está el fragmento casi construido, sólo nos queda modificar la cabecera para que se vea claramente que es el primer fragmento.

Activamos el flag *MF*.

TL = (IHL x 4) + (NFB x 8).

Calculamos un *checksum* para este fragmento, una vez modificada la cabecera.

Como en *OTL* hemos guardado la longitud total del datagrama original, podemos sobrescribir el valor de la variable *TL* sin perder ese dato, que más adelante nos hará falta. El nuevo *TL* (para este fragmento) se calcula en función de la cabecera (*IHL*) y de los datos (*NFB*).

Paso 6 –

El primer fragmento ya está listo, y se envía al nivel de enlace. Continuamos ahora con el segundo fragmento.

Paso 7 –

Copiamos la cabecera del datagrama original al nuevo fragmento. Habrá que revisar el campo *Opciones* de la cabecera IP, para ver cuáles de las opciones tendrán que ser copiadas a todos los fragmentos. Por tanto, la cabecera del fragmento podría ser más pequeña que la del datagrama original, al no tener copia de todas las opciones.

Paso 8 –

Como este algoritmo sólo explica como dividir en 2 fragmentos, asumimos que éste es el último, por lo que añadimos después de la cabecera todos los datos que faltaban (los que había en el datagrama original a partir de la posición *NFB x 8*).

Paso 9 –

Ahora ajustamos varios campos de la cabecera del fragmento, en 5 pasos:

IHL = ((OIHL x 4) – L + 3) / 4.
TL = OTL – (NFB x 8) – ((OIHL – IHL) x 4).
FO = OFO + NFB.
MF = OMF.

Calculamos el *checksum*. Vaya formulitas, ¿eh?

En primer lugar, vemos que el *IHL* del nuevo fragmento, como dijimos, podría ser menor que el del datagrama original, porque podría haber opciones que no se han copiado. La variable *L* que he puesto en la fórmula es la *longitud de las opciones que no han sido copiadas en el fragmento.*

En el caso del *TL* del fragmento, lo obtenemos a partir del *TL* del datagrama original, restándole todo lo que ocupaba el primer fragmento, y la diferencia del tamaño de las cabeceras. Por supuesto, si sólo hay dos fragmentos, el tamaño del segundo fragmento será el tamaño del original sin fragmentar menos el tamaño del primer fragmento. Con respecto a *FO*, es evidente que, ya que el *FO* del primer fragmento era 0, el del segundo dependerá únicamente de *NFB*, que es lo que ocupaban los datos del primer fragmento.

Por último, el flag *MF* estará activo sólo si el propio datagrama original sin fragmentar era ya un fragmento de alguna fragmentación anterior.

Paso 10 –

El fragmento ya está listo, y lo podemos enviar a la capa de enlace para que lo procese. **FIN.**

Bueno, bueno, bueno... Os dije que esto no iba a ser fácil.

Pero llevamos ya el suficiente tiempo dando caña a los protocolos como para que os pueda exigir ya a estas alturas que os lo curréis bien y analicéis paso a paso el algoritmo para comprenderlo. Es difícil darlo ya más masticado de lo que os lo he dado yo, así que mucho más no puedo hacer. Lo que es complicado, es complicado.

2.2. Algoritmo de reensamblado de datagramas del RFC 791.

Este algoritmo es bastante más denso y complicado de entender, así que en este caso es casi mejor contar con la descripción general del algoritmo que viene en el RFC antes de detallar sus pasos.

Esta descripción nos aclara algunos detalles que quizá nos quedaban como lagunas hasta ahora. Por ejemplo, el mecanismo para **diferenciar un datagrama fragmentado de uno no fragmentado**, es que los **no fragmentados** siempre tendrán su **FO** (Fragment Offset) **a 0**, y al mismo tiempo el **flag MF** (More Fragments) **a 0** también.

Si, en cambio, su FO no fuese 0, pero sí el MF, entonces sabríamos que se trata del **último fragmento**. Si el FO si fuese 0, pero el MF no, entonces tendríamos el **primer fragmento.** Por último, si ni el FO ni el MF son 0, entonces se trataría de un **fragmento intermedio** (ni el primero ni el último). Resumo todo esto en una tabla:

FO	MF	Tipo de datagrama
0	0	*No fragmentado.*
0	1	*Primer fragmento de un datagrama.*
>0	0	*Último fragmento de un datagrama.*
>0	1	*Fragmento intermedio de un datagrama.*

Sabiendo esto, lo primero que se hace cuando se recibe un datagrama es comprobar estos dos campos. **Si FO y MF están a cero**, entonces no se trata de un fragmento, y **no hay que reensamblar.**

En cambio, si alguno de los dos no es cero, tenemos que ver **a qué datagrama** corresponde ese fragmento. Ya sabemos que los datagramas se identifican por los campos de la cabecera IP: **dirección IP de origen, dirección IP de destino, protocolo, e identificador**. A partir de estos datos, identificamos el datagrama.

Ahora se nos pueden dar dos situaciones: que sea el primer fragmento que nos llega de ese datagrama, o que antes ya nos hubieran llegado otros fragmentos. Por supuesto, hablo del primer fragmento **EN LLEGAR**, que no tiene por que ser el primer fragmento de los que componen el datagrama. Los fragmentos pueden llegar al destino en cualquier orden, por diversos motivos. No siempre el primero que recibimos es el que tiene *FO = 0*, y *MF = 1* (es decir, el primer fragmento que compone el datagrama original).

En el caso de que sea el primer fragmento **QUE NOS LLEGA**, tendremos que reservar recursos en nuestra máquina para preparar el advenimiento del nuevo datagrama (qué religioso me ha quedado eso).

Son varios los **recursos** que hay que reservar. En primer lugar, por supuesto, un **buffer** (espacio de memoria) en el que iremos componiendo los datos que forman el datagrama según nos van llegando en desorden. El buffer será como una mesa vacía sobre la que iremos colocando las piezas de un puzzle según las vayamos sacando de la caja.

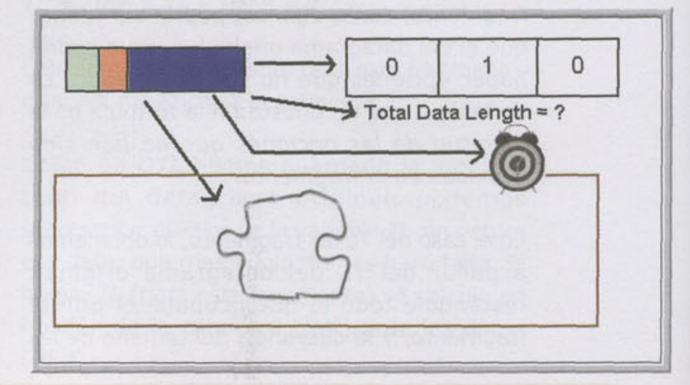
En realidad no se trata sólo de un buffer, si no de dos: uno para los **datos,** y otro para la **cabecera.** Aparte de los buffers, se crea también una **tabla** vacía donde se van marcando los fragmentos según se van recibiendo (una tabla que simplemente contiene **un bit por cada fragmento**, marcando a **1** los que vayan llegando). También tenemos una variable que inicializamos a cero, que es el **Total Data Length**, que ahora veremos en qué consiste. Por último, tenemos un **temporizador** que nos servirá para no permanecer eternamente esperando fragmentos que se hayan podido perder por el camino.

Si el temporizador llega al fin de su cuenta y no se han recibido todos los fragmentos, entonces habrá que descartar el datagrama incompleto, y liberar los recursos.

Vamos a ver un ejemplo de como se va reensamblando un datagrama.

En primer lugar, recibimos un fragmento con **Offset 3000**. Nosotros aún no lo sabemos, pero éste será el *tercero* de los fragmentos que componen el datagrama.

Lo que sí que sabemos es que es un **fragmento intermedio**, ya que su bit **MF** está activado, y su **Offset** es mayor que 0.



En esta imagen podemos ver cómo la recepción del datagrama, al ser el **primero EN LLEGAR** que tenga esa identificación (es decir, esa combinación de **ip de origen, ip de destino, protocolo, e identificador**), causa la creación de una serie de recursos.

Arriba a la derecha podemos ver la **tabla** binaria en la que se van marcando los fragmentos según van llegando. A priori no sabemos el tamaño que tendrá esa tabla, ya que no sabemos cuántos fragmentos habrá, pero lo que sí que sabemos es que en el momento en que todas las posiciones de la tabla estén a 1, el datagrama estará completo.

Lo que sí que sabemos es que, al ser un fragmento intermedio, como mínimo tiene que haber un fragmento a su izquierda, y otro a su derecha, por lo que podemos crear 3 posiciones en la tabla, de las cuales sólo está ocupada la intermedia.

Debajo de la tabla tenemos la variable **Total Data Length**, cuyo valor de momento desconocemos, así que la inicializamos a 0, o con cualquier valor.

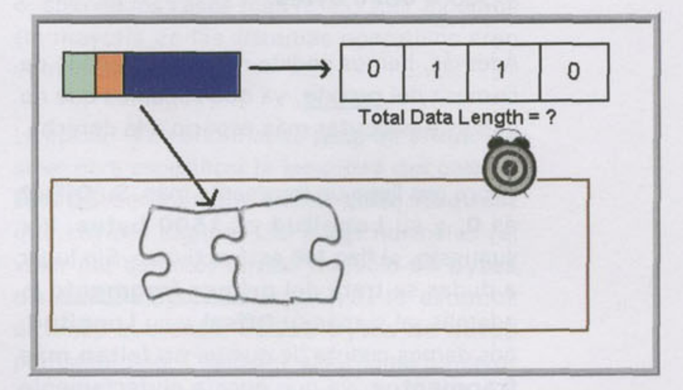
Debajo tenemos el **temporizador,** cuya cuenta se pone en marcha para no quedarnos esperando eternamente el resto del datagrama.

Por última, debajo tenemos la mesa marrón sobre la que iremos construyendo el **puzzle.** No sabemos de antemano qué tamaño tendrá, así que procuramos hacerla bastante grande para que nos quepan todas las piezas. Esta mesa es el **buffer** de recepción.

La primera pieza del **puzzle** se saca de los datos del fragmento que acabamos de recibir.

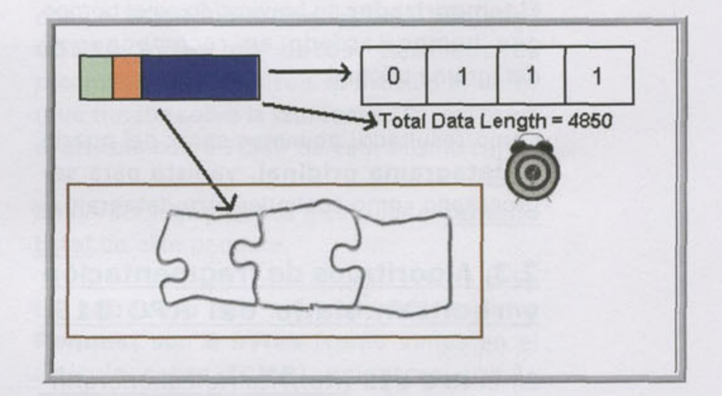
A continuación, nos llega otro fragmento, con **Offset 1500**. Nosotros no lo sabemos, pero este fragmento es el *segundo* de los que forman el datagrama original. Como su **Offset** sigue sin ser **0,** sabemos que sigue tratándose de un **fragmento intermedio**. Por supuesto, el flag **MF** tiene que estar activado por narices, ya que el fragmento que habíamos recibido previamente está a continuación de éste.

Fijándonos en la **Longitud** de este segundo fragmento, vemos que mide **1500 bytes**, por lo que encaja en nuestro **puzzle** perfectamente pegado justo "a la izquierda" del fragmento que recibimos antes.



Este fragmento modifica la **tabla,** añadiendo un **1** en una posición intermedia, quedando ahora la tabla con 4 casillas. Además, añade una nueva pieza al **puzzle**.

Ahora nos llega un fragmento con **Offset 4500**, y con el **flag MF a 0.** Esto significa que es el **último fragmento**. Como la **Longitud** del primer fragmento que nos llegó era de **1500 bytes**, al **sumar su offset y su longitud** tenemos: 3000 + 1500 = **4500**, que coincide con el Offset de este último fragmento. Por tanto, el último fragmento está en nuestro **puzzle** justo "a la derecha" del primer fragmento que nos llegó.

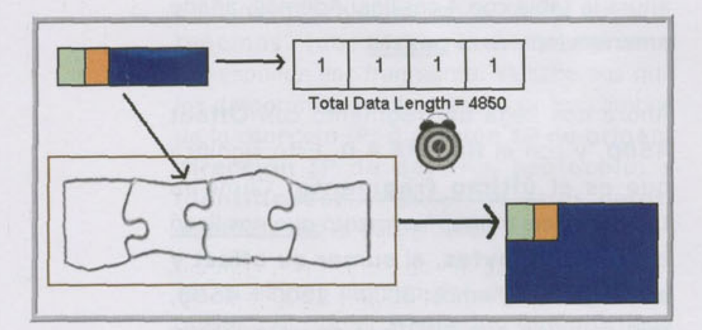


Nuestra nueva **tabla** ya no tendrá hueco por la derecha, por lo que sólo nos falta rellenar los fragmentos que nos falten por la izquierda.

La variable **Total Data Length** ahora puede ser calculada. Cogemos el **Offset** de nuestro fragmento, y **le sumamos su Longitud**. En este ejemplo la **Longitud** de este último fragmento es de **350 bytes**, por lo que: 4500 + 350 = **4850 bytes**.

Además, hemos podido reducir el tamaño de la mesa del **puzzle**, ya que sabemos que no vamos a necesitar más espacio a la derecha.

Ahora nos llega un fragmento más. Su **Offset** es **0**, y su **Longitud** es **1500 bytes**. Por supuesto, el flag **MF** está activado. Sin lugar a dudas se trata del **primer fragmento** y, además, al sumar su **Offset** y su **Longitud**, nos damos cuenta de que ya **no faltan más fragmentos**, ya que encaja perfectamente en el **puzzle** "a la izquierda" del que ahora ya sabemos que es el segundo fragmento.



Como vemos, la **tabla** ha sido ya completada.

El **puzzle** está completo con todas sus piezas.

El **temporizador** no ha vencido en el tiempo que hemos tardado en recomponer el datagrama original.

Como resultado, podemos sacar del puzzle el **datagrama original**, ya listo para ser procesado como cualquier otro datagrama.

2.3. Algoritmos de fragmentación y reensamblado del RFC 815.

En el **RFC 815** (ftp://ftp.rfc-editor.org/in-notes/rfc815.txt) hay detallados unos algoritmos alternativos que simplifican bastante los algoritmos del RFC 791. Una de las grandes ventajas es que el algoritmo de reensamblado del RFC 815 consume menos recursos, lo cual veremos más adelante que puede suponernos una ayuda contra ciertos tipos de ataques, aunque en cualquier caso si se proponen hacernos un DoS con alguna de las técnicas que explicaremos, de poco nos va a servir ahorrarnos unos cuantos bytes en los buffers...

Os dejo como ejercicio empollaros este RFC, que es muy cortito, a ver si conseguís comprender más o menos los algoritmos que explica.

3. Problemas de seguridad de la fragmentación de datagramas.

Llegamos ya a la parte más divertida. 

Los ataques que aprovechan la fragmentación IP han estado siempre muy extendidos, y esto ha dado lugar incluso a la aparición de un par de RFCs tratando exclusivamente este problema.

El RFC que trata algunos de estos problemas de forma más general es el **RFC 1858**. El **RFC 3128** trata sobre una técnica concreta de explotación, y la protección contra la misma, ampliando así la información del RFC 1858.

Pero no nos podemos conformar con la información contenida en estos RFCs, ya que no todos los posibles ataques de fragmentación existentes están detallados en estos documentos. De hecho, algunos de los más conocidos, como el **Ping of Death**, o el **Teardrop** no son mencionados en ningún documento RFC (o al menos que yo sepa).

A grandes rasgos, los ataques que explotan la fragmentación se pueden dividir en dos grupos: ataques DoS, y ataques para atravesar firewalls.

3.1. Ataques DoS

Estos son los ataques que considero más fáciles de comprender en principio.

Los **algoritmos de reensamblaje** de fragmentos especificados en los **RFCs 791 y**

815 son propensos a gran cantidad de problemas ya que, como ocurre con la mayoría de los protocolos básicos de Internet, fueron pensados para situaciones de buena fe, donde todo funciona como debería.

Por tanto, no se tienen en cuenta muchas situaciones anómalas, como valores imposibles para los offsets de los *fragmentos*, flujos de datagramas incompletos, o fragmentos que forman datagramas demasiado grandes.

Vamos a ver en detalle algunas de las técnicas DoS que permiten explotar este exceso de confianza de los algoritmos de reensamblado. Los sistemas operativos modernos están "protegidos" contra muchas de estas técnicas, pero aún siguen quedando servidores sin actualizar que pueden ser vulnerables. En cualquier caso, siempre es interesante conocer todo lo que se puede hacer gracias a un algoritmo mal diseñado.

Estad preparados, porque aquí hay mucha chicha.

3.1.1. Ping of Death

Este ataque, el ping de la muerte, fue muy popular cuando fue "descubierto" en 1996.

Durante un par de años fue el paraíso sobre todo para los *scriptkiddies* (o *lamers*, para que nos entendamos) que encontraron una forma "universal" de destruir cualquier máquina con sólo escribir un simple comando.

Este comando se podía escribir directamente desde **Windows 95 o Windows NT**, que eran los sistemas operativos de Microsoft que había en el momento, sin necesidad ni siquiera de instalar ninguna aplicación para explotar el ataque. En cambio, para los usuarios de **Linux** era necesario compilar un pequeño código.

Si un usuario de Windows 95/NT quería petar a un tío cuya ip fuese *80.12.1.4*, sólo tenía que escribir en una ventana **MS-DOS**:

ping -l 65510 80.12.1.4

y con eso: hop! la máquina *80.12.1.4* quedaba petada.

Dependiendo del sistema operativo de la víctima, la máquina podía reiniciarse, colgarse, o, sólo en los casos más raros, salir indemne (la mayoría de los sistemas operativos eran vulnerables).

La opción **-l** en el comando **ping** de **Windows** sirve para especificar la **longitud del campo DATOS** del mensaje **ICMP Echo Request** que enviará el ping. Los pings normales (el valor por defecto) envían tan sólo **64 bytes de datos**. Con esta opción **-l** le estamos diciendo que envíe **65510 bytes de datos** por cada ping. Cualquier valor igual o mayor que 65510 puede servir para llevar a cabo el DoS. Más adelante veremos por qué esto causa un problema.

En el caso de **Linux,** la longitud del campo de datos se especifica con la opción **-s,** pero en este caso está limitado a un máximo de **65507 bytes**, lo cual no permite llevar acabo este ataque (si, esos 3 bytes son fundamentales, como ya veremos).

Hoy día dudo que queden muchas máquinas conectadas que todavía sean vulnerables al Ping of Death, pero me parece un ejemplo muy didáctico de en qué puede consistir un ataque DoS que explote la fragmentación IP.

Vamos a ver paso a paso qué es lo que pasa con un **ICMP Echo Request** que contenga **65510 bytes de datos**. Para ello os recomiendo que reviséis el artículo anterior (que trataba sobre la **cabecera IP**), así como el artículo sobre **ICMP** de este mismo curso.

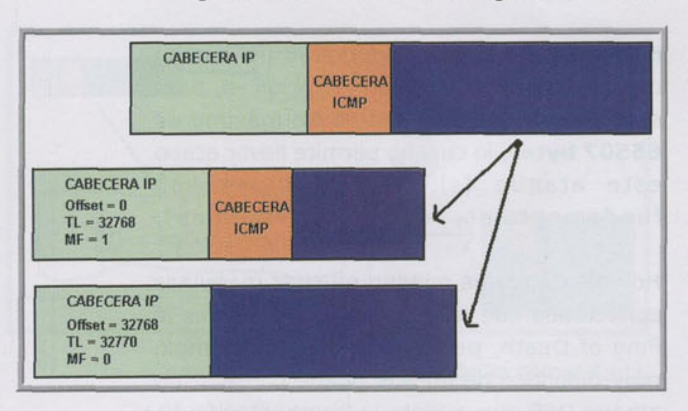
En primer lugar, vamos a calcular el **<u>tamaño total</u>** de este paquete.

La **cabecera** del mensaje **ICMP Echo Request** son **8 bytes** (como vimos en el artículo sobre **ICMP**), mientras que la **cabecera IP** sin opciones es de **20 bytes**. Por tantos, si sumamos estos **28 bytes** de cabeceras a los **65510 bytes** del campo de datos, tenemos un total de **65538 bytes**.

¿Por qué el ping de algunos sistemas operativos como Linux no permiten hacer este ping? Pues porque, si recordamos del artículo anterior el campo **Total Length** de la cabecera IP, sabremos que este campo especifica con *sólo* **16 bits** el tamaño del datagrama, por lo que el tamaño máximo de un datagrama tendría que ser **65536 bytes**. Precisamente en este punto es donde está la clave del ataque.

Sólo hay una forma en la que este datagrama puede enviarse, ya que el campo **Total Length** de la cabecera IP no da más de si. El único medio es **fragmentar** el datagrama, y ajustar de forma adecuada el tamaño de cada fragmento.

En el caso más sencillo, el datagrama sería fragmentado en dos de la siguiente forma:



Estos fragmentos siguen siendo demasiado grandes para pasar por cualquier red, por lo que sin duda serían fragmentados a su vez para ajustarse a la **MTU** de la red por la que tuviesen que circular. Pero esto lo vamos a obviar, por lo que supondremos que lo que llega a la víctima son directamente estos dos paquetes (para simplificar mucho el ejemplo).

El sistema operativo de la víctima recibirá el primer fragmento, con **Offset 0, tamaño de paquete (TL) de 32768 bytes**, y el flag **MF** activado. A continuación (aunque no tendrían por qué llegar necesariamente en este orden, como ya sabemos) recibirá el segundo paquete, con **Offset 32768**, **tamaño de paquete de 32770 bytes**, y el flag MF desactivado (ya que es el último fragmento).

De momento, ningún problema: cada paquete es correcto por separado. Pero el problema viene cuando el sistema trata de **reensamblar** estos dos fragmentos. Al unir los fragmentos el sistema ha de computar el tamaño del datagrama original (el **Total Length**). Para ello se hace una **suma de 16 bits** de los **tamaños** de cada fragmento. El resultado de esta suma en este caso sería 32768 + 32770 = **65538**, lo cual se sale del rango de la suma, que estaba limitada a 16 bits.

Esta suma fuera de rango no estaba prevista por el sistema operativo y, por tanto, producirá resultados inesperados que, en la mayoría de los casos, daban lugar a que el sistema se colgase, o bien se reiniciase.

¿Por qué se utilizaba precisamente un **ping** cuando cualquier otro tipo de datagrama habría servido para el mismo fin? Pues bien, en efecto hay variantes del **POD** (**P**ing **O**f **D**eath) que utilizan otro tipo de datagramas, como datagramas **UDP**, u otros tipos de **ICMP**. Pero las grandes ventajas de usar **ping** son dos: por una parte, que **ICMP** es un servicio **no orientado a conexión**, por lo que no es necesaria ninguna información para colarse en una conexión como podría ser el caso de **TCP**. El hecho de no tener que colarse en una conexión nos da la gran ventaja de que podemos enviar el ataque sin preocuparnos de tener que recibir ninguna respuesta. Al no tener que esperar respuestas, podemos perfectamente enviar el ataque con un **IP-Spoofing**, tal y como veremos en el próximo artículo, en el cual entraremos en todo detalle en las técnicas de IP-Spoofing. Esto permitía realizar el ataque de forma totalmente anónima.

La otra ventaja de utilizar el ping es que, hasta el momento en que surgieron este tipo de ataques, prácticamente ningún **firewall** filtraba los mensajes **ICMP Echo Request**, al considerarlos totalmente inofensivos.

Después de la aparición de estos ataques basados en el ping, la mayoría de los firewalls empezaron a filtrar los mensajes **ICMP Echo Request**. Esto fue una medida de defensa

bastante ingenua, ya que enseguida surgieron nuevas variantes del **POD** que utilizaban cualquier otro tipo de *datagramas*, ya que el fin último al fin y al cabo era simplemente conseguir hacer llegar un datagrama fragmentado, fuese de la naturaleza que fuese.

¿Cómo puede ser entonces que hoy día la *inmensa mayoría* de los sistemas sean inmunes a este tipo de ataques, si no se pueden evitar filtrando el ping en un firewall?

Pues sencillamente, la solución al problema se implementó en el propio **sistema operativo**. *Salieron parches* para los sistemas existentes en el momento, y además todos los sistemas que se hicieron después eran *ya inmunes "de fábrica"*, es decir, tenían prevista la situación de que un datagrama pudiese medir más de 65536 bytes tras ser reensamblados sus fragmentos. Como ejemplo, los kernels de **Linux** posteriores al **2.0.23** son ya todos inmunes a estos ataques.

En cualquier caso, no sólo nos tenemos que resignar a confiar en lo que haga nuestro sistema operativo, si no que también podemos nosotros tomar parte activa en la protección de nuestra máquina ante estos ataques.

A la hora de implementar protecciones en nuestro firewall contra ataques específicos es fundamental conocer con detalle el funcionamiento del ataque.

Por ejemplo, buscando por Google, podemos encontrar esto:

```
# Block PING OF DEATH
$IPT -A FORWARD -p icmp --icmp-type echo-request -m limit --limit 1/s -j LOG --log-prefix "Ping of Death Blocked: "
$IPT -A FORWARD -p icmp --icmp-type echo-request -m limit --limit 1/s -j DROP
$IPT -A INPUT -p icmp --icmp-type echo-request -m limit --limit 1/s -j LOG --log-prefix "Ping of Death Blocked: "
$IPT -A INPUT -p icmp --icmp-type echo-request -m limit --limit 1/s -j DROP
```

Si habéis seguido el **curso de implementación de firewalls** de la revista, *sabréis que estas líneas son* una serie de reglas de **IPTABLES,** el firewall implementado en el kernel de **Linux.** Si conocemos el *funcionamiento de iptables* (**man iptables**) sabremos que lo que hacen estas líneas es limitar a que sólo podamos recibir un máximo de **un ping por segundo** (*protocolo ICMP, tipo de mensaje Echo Request, y limite de 1/segundo*).

Aunque estas reglas de iptables presuman de ser una protección contra el **POD**, nada más lejos de la realidad. Lo único que hacen es "evitar" un **flood** de pings, que podría ser resultado de algún ataque **DDoS** (DoS distribuido).

Lo que os quiero decir con esto es que no podemos fiarnos de las soluciones que encontremos por ahí, ya que si bien muchas veces encontraremos soluciones adecuadas, en cualquier caso siempre tenemos que estar preparados para **analizar con nuestros propios conocimientos** si esa es realmente una solución para nuestro problema, o si más bien el tío que lo escribió (probablemente con toda su buena fe) no tenía mucha idea de en qué consiste realmente el ataque.

Seguramente alguno de vosotros en este punto habrá pensado que, si bien esa protección no sirve contra el **POD**, si que podría servir contra un ataque **Smurf**, *que consiste* precisamente en un **flood de pings** (tal y como expliqué en el artículo sobre **ICMP**).

Pues para todos los que hayáis pensado eso, os pongo un punto positivo, ya que demuestra que habéis estado bastante atentos al curso.

Ahora bien... para quien pongo ya no uno sino tres puntos positivos es para el que se haya dado cuenta de que los que hayan pensado eso están equivocados, ya que falla un pequeño detalle. El ataque **smurf,**

en efecto, se lanza mediante pings, pero lo que la víctima recibe no son los pings, si no los **pongs,** es decir, no recibe un **flood** de Echo Request, si no de **Echo Reply**. Por tanto, esta sí sería una **protección contra Smurf con iptables**:

```
# Bloqueo de Flood por ataque Smurf
$IPT -A FORWARD -p icmp --icmp-type echo-reply -m limit --limit 1/s -j LOG --log-prefix "Smurf a red interna: "
$IPT -A FORWARD -p icmp --icmp-type echo-reply -m limit --limit 1/s -j DROP
$IPT -A INPUT -p icmp --icmp-type echo-reply -m limit --limit 1/s -j LOG --log-prefix "Smurf al firewall: "
$IPT -A INPUT -p icmp --icmp-type echo-reply -m limit --limit 1/s -j DROP
```

Esto, por supuesto, no nos serviría para evitar convertirnos en un **amplificador Smurf**, es decir, convertirnos en colaboradores involuntarios del ataque contra otra persona.

Como ya vimos, el ataque Smurf depende de máquinas "inocentes" que respondan a los mensajes de ping a la **dirección broadcast** de la red.

La solución a este problema es mejor implementarla a otro nivel, que no es el de las iptables, diciéndole directamente a nuestro sistema operativo que ignore cualquier **ICMP Echo Request** que tenga como destino la **dirección broadcast** de la red. Esto lo podemos hacer con la siguiente línea:

```
echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

Esta línea la podemos incluir como parte de los scripts de arranque del sistema (los clásicos de **/etc/rc.d**), o bien en el archivo **/etc/sysctl.conf**.

Pero no nos salgamos del tema, y pensemos cómo podemos protegernos contra el **POD** con iptables. Una solución específica contra **POD** sería esta:

```
# Esto sí que bloquea un Ping Of Death
iptables -A INPUT -p icmp --icmp-type echo-request --fragment -j LOG --log-prefix "Ping fragmentado: "
iptables -A INPUT -p icmp --icmp-type echo-request --fragment -j DROP
iptables -A FORWARD -p icmp --icmp-type echo-request --fragment -j LOG --log-prefix "Ping fragmentado: "
iptables -A FORWARD -p icmp --icmp-type echo-request --fragment -j DROP
```

Aquí estamos rechazando cualquier fragmento que pertenezca a un datagrama **ICMP Echo Request**. No hay ningún motivo "legal" por el que nadie quisiese enviarnos un Echo Request fragmentado, por lo que no estamos impidiendo el correcto funcionamiento de nuestra red por añadir esta regla.

Como podemos suponer, esta regla no es lo suficientemente restrictiva, ya que si bien evitaría el ataque **POD,** alguna de sus **variantes** (las que utilicen otros tipos de mensaje ICMP, o bien otros protocolos, como UDP o TCP) sí que pasarían por el firewall.

Para evitar las variantes que utilicen otro tipo de mensajes ICMP podríamos ser menos específicos en la regla, y **filtrar cualquier ICMP fragmentado**, ya que en general no existe ningún uso "legal" para ningún tipo de ICMP fragmentado:

```
# Esto bloquea cualquier ataque por mensajes ICMP fragmentados
iptables -A INPUT -p icmp --fragment -j LOG --log-prefix "ICMP fragmentado: "
iptables -A INPUT -p icmp --fragment -j DROP
iptables -A FORWARD -p icmp --fragment -j LOG --log-prefix "ICMP fragmentado: "
iptables -A FORWARD -p icmp --fragment -j DROP
```

Con esto aún no solucionaríamos el problema de los ataques tipo **POD** que utilicen otros protocolos distintos a ICMP. Por lógica podríamos pensar que otros protocolos, como **TCP**, que transportan protocolos de nivel superior que pueden llevar un gran flujo de datos (como **FTP**), no podrían funcionar correctamente sin la fragmentación.

Por tanto, de momento dejamos aquí el tema, y nos conformamos con haber encontrado una solución para que nuestro firewall filtre los ataques **POD** clásicos (con mensajes **ICMP**), y más adelante iremos viendo cómo solucionar el problema con el resto de protocolos.

3.1.2. Jolt y jolt2

Estos dos **exploits** explotan dos bugs diferentes, pero ambos relacionados con la fragmentación.

Jolt es más antiguo, y os costará encontrar alguna máquina vulnerable hoy día, pero **Jolt2** se supone que puede tirar a varios sistemas, incluido **Windows 2000**.

No hay mucho que explicar sobre estos exploits, ya que lo que hacen simplemente es enviar una serie de paquetes fragmentados que muchos sistemas operativos no saben cómo tratar y, por tanto, pueden colgar el sistema.

¿Y por qué no hay mucho que contar? Pues porque son unas técnicas más empíricas que lógicas. Al menos hasta donde yo he podido documentarme, no he encontrado ninguna explicación convincente de por qué estos paquetes pueden colgar un sistema. De hecho, lo que sí que he encontrado han sido discusiones en foros de seguridad sobre qué hace o deja de hacer cada uno, y por qué lo hace o lo deja de hacer.

En cualquier caso, sea magia o sea ciencia, el caso es que ambos exploits han sido de los más famosos en la historia de los **nukes**. Como ya conté en otro artículo, existen básicamente dos tipos de **DoS**: el **flood** y el **nuke**.

El **flood** consiste en tirar un sistema a base de saturar sus recursos por fuerza bruta, mientras que el **nuke** consiste en explotar un bug concreto de un sistema que no es capaz de responder bien ante cierta situación.

Por ejemplo, el **POD** es un claro ejemplo de **nuke**, en el cual con un sólo datagrama podemos tirar un sistema. En cambio, el **smurf** es un claro ejemplo de **flood,** en el cual se tira el sistema a base de saturarlo con miles de ICMPs.

Con respecto a **jolt2** hubo ciertas discusiones sobre si realmente era un nuke, o si en realidad era un simple flood, ya que el exploit que circulaba (***jolt2.c***, lo podéis buscar en **Google**) enviaba paquetes a toda pastilla, tanto incluso que saturaba la conexión del propio atacante. En cambio, poco después circuló una nueva versión de jolt2 (***jolt2mod.c***, también lo podéis buscar) que ralentizaba el envío de paquetes para no saturar la conexión del atacante.

Cada vez que hablo de un tema diferente me gusta contarlo de una forma diferente, así que os voy a hablar en primer lugar de los **retoques** que he tenido que hacer al **código fuente** de ambos exploits.

En primer lugar, al tratar de compilar el código de **Jolt** (***jolt.c***, lo podéis buscar también), con:

gcc jolt.c -o jolt

Descubro que no compila, ya que da un error con un campo de una estructura que no existe. Así que edité el código:

vi jolt.c

Y busqué cualquier referencia a ese campo, y vi que aparecía una única vez, en una línea que sólo le daba un valor **0.** Así que comenté esa línea (la metí entre **/*** y ***/**), salvé el archivo, y lo volví a compilar, esta vez sin problemas.

No os he dicho los pasos exactos para proponeros que lo hagáis vosotros mismos como ejercicio.

Para usar el exploit sólo tenéis que decir la ip de destino y la de origen (permite **ip-spoofing**):

./jolt ip.victima ip.spoofeada

Lo más probable es que esta técnica no os funcione, ya no sólo porque queden pocos sistemas vulnerables, si no porque probablemente ni siquiera pueda atravesar alguno de los routers que haya en la ruta, al ser un paquete malformado que podría ser rechazado.

Aún así, por si tenéis curiosidad, podéis analizar con un **sniffer** lo que hace Jolt. Os ahorro el trabajo si os cuento que simplemente envía una serie de **fragmentos solapados**, es decir, fragmentos en los que el campo **offset** de un fragmento no se corresponde con el **total length** del anterior fragmento. Más adelante veremos usos mucho más interesantes para el solapamiento de fragmentos.

Con respecto a **Jolt2**, también tuve que hacer una modificación al código fuente. En este caso usé la variante **jolt2mod.c**.

Supuestamente, el exploit debería permitir un **ip-spoofing** utilizando la opción **-s**, tal y como podéis ver si después de compilarlo:

gcc jolt2mod.c -o jolt2mod

Lo ejecutáis, para que os muestre su funcionamiento:

```
Bellatrix:/home/pyc/hackxcrack/ip3# ./jolt2mod
Usage: ./jolt2mod [-s src_addr] [-p port] dest_addr
Note: UDP used if a port is specified, otherwise ICMP
```

En cambio, al analizar el funcionamiento de jolt2 con un **sniffer** (*Ethereal*, concretamente) descubrí que hace caso omiso a la opción **-s**. Revisando el código fuente descubrí que en una línea en la que debería poner ***src_addr*** ponía ***dst_addr,*** por lo que lo que se estaba cambiando no era la ip de origen (para hacer el ip-spoofing), si no la ip de destino. Como la ip de destino (*dst_addr*) en cualquier caso se volvía a modificar más adelante, el efecto de esas líneas era nulo. Una vez corregido el código es así como queda esa parte:

```
src_addr = host_to_ip(optarg);
if (!src_addr)
    quit("Bad source address given.");
break;
```

Os dejo como ejercicio también que os las apañéis vosotros para hacer esta corrección.

Una vez corregido esto, vemos el mismo problema. Los paquetes están malformados y es probable que no puedan pasar por todos los routers. Si analizamos su funcionamiento con un **sniffer** (o analizando el **código fuente**) veremos que lo que hace es enviar una y otra vez el mismo fragmento, con todos los campos exactamente iguales. ¿Por qué esto puede tirar un Windows? Sinceramente, no tengo ni idea; eso que se lo pregunten a Bill Gates.

3.1.3. Ataques de flood de fragmentos

Si vuestras mentes están despiertas probablemente se os haya ocurrido pensar que **Jolt2** funciona porque cada vez que Windows recibe un fragmento **reserva unos recursos** para ese paquete (como ya vimos antes), y al no recibir el resto de fragmentos llega un momento en el que los recursos del sistema se agotan.

Esta es una deducción muy buena, pero tiene un pequeño fallo, y es que el **identificador** es el mismo para todos los fragmentos. Por tanto, cada fragmento nuevo no obligaría al sistema (en teoría) a reservar nuevos recursos.

En cambio, si utilizamos **diferentes identificadores** para cada fragmento, sí que podríamos llegar a saturar los recursos de la máquina, al hacer reservar **buffers** para cada nuevo fragmento, esperando todos ellos llegar a completar el datagrama al que pertenecen.

Esto es algo que "fácilmente" puede ocurrir con firewalls que ignoren los fragmentos, incluso de forma no intencionada.

Un comportamiento típico de un **firewall** ante un datagrama fragmentado es analizar únicamente el **primer fragmento** (que es donde se encuentran las cabeceras). En caso de que ese primer fragmento sea rechazado por el firewall, el resto de fragmentos **sí** que atravesarán el firewall en cualquier caso. Por tanto, a la máquina que haya detrás le llegarán todos los fragmentos excepto el primero. Esto provocará que la máquina reserve recursos inútilmente para formar un datagrama que nunca llegará, ya que le falta el primer fragmento.

3.1.4. Teardrop

Este exploit de tipo **nuke** utiliza también el (igual que **Jolt**), pero en este caso sí que soy capaz de explicar mejor por qué funciona.

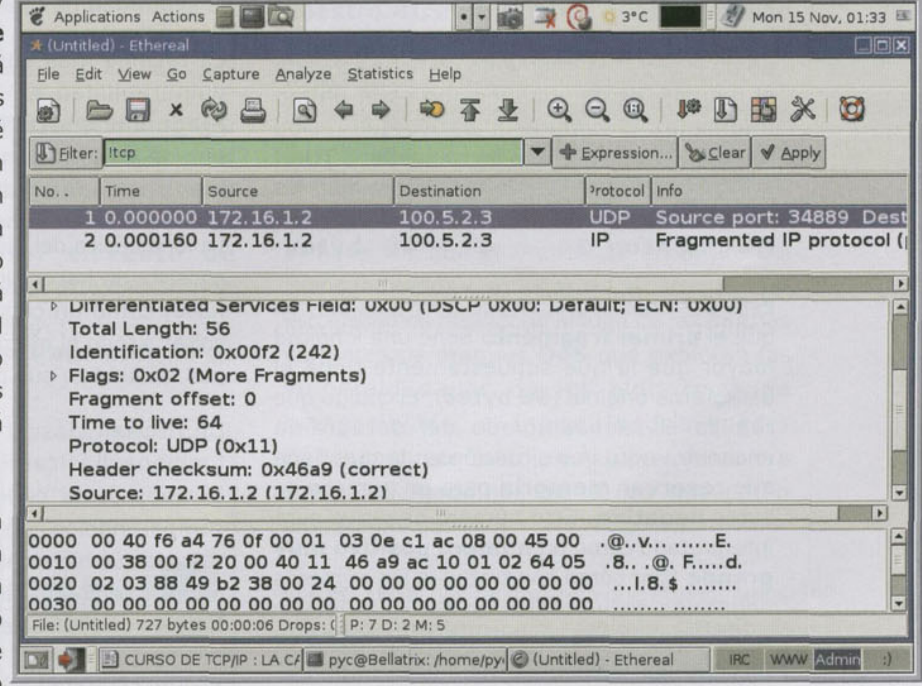
En mi defensa puedo decir que **Jolt** es un exploit para **Windows**, y como el **código fuente** de windows no está disponible, muchas veces es imposible decir por qué está fallando algo. En cambio, **Teardrop** es un exploit que afecta también a **Linux,** cuyo código fuente está disponible para todo el mundo. Por tanto, viendo el código, si que podemos decir **por qué** está fallando algo.

Aquí llegamos a una cuestión que se plantea muchas veces a la hora de discutir si es bueno o no tener el código fuente disponible. Un argumento bastante irreflexivo consiste en decir rápidamente: *"pero claro, si un hacker maligno puede ver el código fuente, podrá entonces encontrar formas de atacar a ese sistema".*

Pues sí, en efecto, revisando el código fuente de un sistema operativo puedes encontrar problemas potenciales de seguridad, pero... ¿acaso es necesario eso? ¿Acaso no surgen constantemente nuevos exploits para Windows, del cual "nadie" tiene el código fuente?

En cambio, el hecho de tener el código fuente disponible sí que favorece todo lo contrario: en el momento en que surja un problema, cualquiera podrá buscar la causa, y darle un remedio. No dependemos de las actualizaciones mensuales de Microsoft (si es que a Microsoft le sale de las narices corregir ese problema). Nosotros mismos podemos corregirlo o, si no, cualquier usuario de la comunidad Linux que amablemente lo haya corregido por nosotros y haya hecho esa solución pública. Una vez más: **GNU - 1**, **Microsoft – 0.**

Bueno, pues volviendo al tema, vamos a razonar el funcionamiento del **Teardrop**. Para ello, vamos a ver una captura de un **sniffer** (*Ethereal*) tras lanzar el **exploit**.



Para ejecutar el exploit, primero tendremos que **compilar** el código teardrop.c:

```
gcc teardrop.c -o teardrop
```

Y a continuación lo lanzamos con:

```
./teardrop ip.spoofeada ip.victima
```

Y vamos al **sniffer** a ver qué ha hecho. Repito una vez más que también se puede analizar el comportamiento mirando el **código fuente** en lugar del sniffer.

Si lo analizamos vemos en primer lugar que sólo son **dos fragmentos** (increíble poder tirar un sistema con sólo dos paquetitos), y que ambos pertenecen al **mismo datagrama** (mismas ips de origen y destino, ambos con protocolo udp, y ambos con el mismo identificador).

Lo que cambia entre los dos fragmentos son los campos **Total Length** y **Offset**, y aquí es donde está la clave.

El **primer fragmento** tiene:
Total Length = 56 bytes
Offset = 0 (flag MF activado)

El **segundo fragmento** tiene:
Total Length = 24
Offset = 24

Como el segundo fragmento no tiene activado el flag MF, es de suponer que es el último fragmento. El sistema operativo que recibe estos fragmentos tratará de computar el **Total Length** del datagrama original sumando los campos **Total Length** y **Offset** del **último fragmento: 24 + 4 = 48 bytes**.

Esto da lugar a una situación imposible, ya que el **primer fragmento** tiene una longitud mayor que la que supuestamente tiene el datagrama original (**56 bytes**). El código que realiza el reensamblado del datagrama encuentra aquí una situación en la que tiene que **reservar memoria** para un número de bytes **negativo**. Este número negativo será interpretado como un **número positivo muy grande** (por cómo son tratados los números negativos en variables en las que no son esperados), por lo que el sistema trataría de reservar demasiada memoria y, por tanto, se colgaría.

Estos cuelgues no ocurren cuando es una aplicación la que trata de reservar demasiada memoria, ya que el sistema operativo se encarga de que esto no ocurra, pero cuando es el propio sistema operativo el que realiza esta operación imposible, entonces no hay santo que nos proteja.

3.2. Saltando firewalls e IDSs.

Vamos a ver aquí un problema de compromiso a la hora de configurar nuestro firewall, ya que muchas veces las soluciones contra los ataques **DoS** favorecen este otro tipo de ataques, y viceversa. Aún así, en el último punto del artículo explicaré la que podría ser una solución definitiva contra ambos tipos de ataques.

Este problema de seguridad es, en mi opinión, mucho más serio y más interesante que el de los ataques DoS. De hecho, es esta cuestión la que dio lugar a la aparición de los **RFCs 1858 y 3128**.

Un ataque **DoS** puede ser un complemento para llevar a cabo un ataque más complejo, como vimos por ejemplo en el caso del **envenenamiento DNS**. En cambio, esta técnica es ya una forma concreta de saltarse la seguridad de una red.

La técnica más conocida que explota esta vulnerabilidad es la conocida como **Tiny Fragment Attack**. Personalmente, me parece una de las formas más elegantes de explotar las potencialidades de un protocolo.

La idea básica del **T**iny **F**ragment **A**ttack (**TFA**) consiste en hacer picar al firewall con un cebo falso, como un caballo de troya, para luego soltar a todo el ejército una vez que el firewall nos ha abierto sus puertas.

Un firewall clásico de filtrado de paquetes funciona básicamente analizando ciertos campos de la cabecera de un paquete, y comparándolos con sus reglas de filtrado. En caso de que haya una coincidencia, ese paquete deberá ser rechazado o aceptado, según lo que indique la regla.

Veamos como ejemplo una lista de reglas de un firewall muy simple:

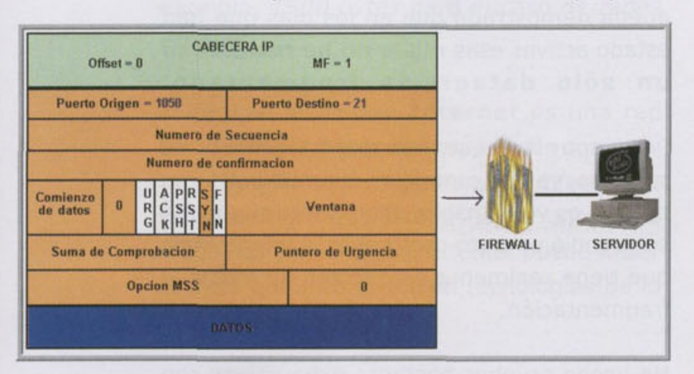
```
Iptables -A FORWARD -p tcp --destport 21 -j ACCEPT
Iptables -A FORWARD -p tcp -j DROP
```

Con estas dos líneas, en primer lugar analizaríamos cualquier paquete **TCP** entrante para ver si el **puerto de destino** es el **21**.

En caso afirmativo, lo **aceptamos**. En cualquier otro caso, el paquete pasaría a la segunda regla, que sencillamente **rechaza** absolutamente todo paquete **TCP** que le llegue.

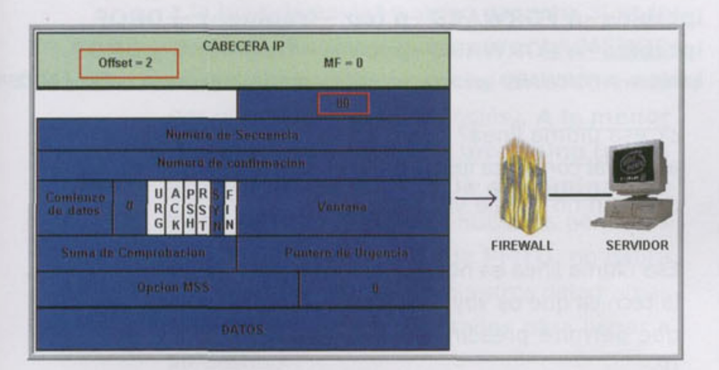
Supongamos que queremos acceder al **puerto 80** de la máquina que hay detrás del firewall. Por supuesto, el firewall nos lo está impidiendo, ya que sólo pasarán los paquetes que tengan como puerto de destino el 21.

Veamos qué ocurre si enviamos este paquete:



Este paquete coincidirá con la primera regla del firewall (**puerto de destino = 21**), por lo que podrá pasar las defensas "inocentemente". Por tanto, el firewall, todo confiado, dejará pasar el **resto de fragmentos** que componen este datagrama.

Pero... ¿y si el **segundo fragmento** es de esta forma? :



Como vemos, el **Offset** es tan sólo de **2 bytes**, por lo que a la hora de reensamblar el supuesto datagrama original, los datos del segundo fragmento se solaparán sobre los del primero :



Como vemos, el datagrama que resulta tras terminar el reensamblado es totalmente diferente del que pasó felizmente por las defensas del firewall, ya que este nuevo datagrama tiene como **puerto de destino el 80**, y no el 21.

Podríamos pensar que una posible solución para este problema podría ser configurar nuestro **firewall** para que él mismo **reensamble** los fragmentos, para aplicar las reglas sobre los fragmentos ya reensamblados. Pero es aquí donde nos damos cuenta de lo que decía al principio de este punto, y es que el remedio podría ser peor que la enfermedad.

Si hacemos que el firewall se encargue del reensamblado, y en lugar de un ataque **TFA** estuviésemos recibiendo alguna de las muchas variantes de ataques **DoS** que explotan las vulnerabilidades de los algoritmos de reensamblado, el firewall podría quedar DoSeado y, por tanto, denegar el servicio a todas las máquinas de la red que estén amparadas por él.

El **RFC 1858** nos recomienda que impongamos un **valor mínimo** para el campo **Offset** de cualquier fragmento (excepto el valor **0**, claro, que sería siempre el Offset del **primer fragmento**), para que un fragmento que no sea el primero no pueda contener nunca parte de la **cabecera TCP**. Es decir, el primer fragmento siempre debe contener, como mínimo, la cabecera completa TCP, y los siguientes fragmentos sólo podrán contener los **datos** del paquete.

En cualquier caso, si continuamos leyendo, encontraremos la que podría ser la solución universal para todos nuestros problemas.

4. Cómo prescindir de la fragmentación, y que todo siga funcionando.

Empecemos preguntándonos, ¿realmente debemos prescindir de la fragmentación?

Pues ya hemos visto muchos motivos de seguridad por los que la fragmentación es poco deseable. Aunque la mayoría de las técnicas que he explicado estén prácticamente obsoletas hoy día, está claro que la fragmentación da mucho juego para encontrar nuevas formas de retorcer los protocolos hasta conseguir que hagan lo que se supone que no deberían hacer.

Por otra parte, el único motivo por el que la fragmentación no es deseable no es sólo la seguridad. También es importante el **consumo extra de recursos** que supone tener que gestionar la fragmentación, y sobre todo el reensamblado. Una máquina que tenga que estar reensamblando datagramas tendrá un aumento en su consumo de CPU y de memoria, dos recursos que en muchas máquinas valen como el oro, y hay que intentar optimizar al máximo.

Después de todo lo que hemos visto, ¿a que os dan ganas de meter estas líneas en las iptables?

iptables -A INPUT --fragment -j DROP
iptables -A FORWARD --fragment -j DROP

Estas líneas rechazarían cualquier paquete fragmentado, fuese de la naturaleza que fuese. Pero claro, ponemos eso, y lógicamente todo dejará de funcionar.... ¿o no?

Para escribir este artículo he tenido que documentarme mucho sobre el tema (como es lógico), y en ningún sitio he leído que nadie aconseje aplicar unas reglas tan radicales a un firewall. De hecho, las reglas que tenía yo en mis propias iptables (cuando me curré mi script de iptables de 358 líneas, hace unos años), aunque trataban el tema de los fragmentos, eran mucho menos restrictivas.

Como para escribir mis artículos no me basta sólo con documentarme, si no que también recurro a la experimentación de todo lo que explico, probé hace unos días a meter estas dos líneas, y... imagia! todo sigue funcionando exactamente igual. No solo eso, si no que mis iptables no han logeado ni un solo fragmento (**iptables -A FORWARD --fragment -j LOG --log-prefix "Fragmento: "**), por lo que queda demostrado que en los días que han estado activas esas reglas **no he recibido ni un sólo datagrama fragmentado**.

Por supuesto, esto no me ha pillado de sorpresa, ya que conocía de antemano el tema del que os voy a hablar ahora, pero sí que me sorprendió un poco comprobar la importancia que tiene realmente esta forma de evitar la fragmentación.

He hecho pruebas bastante exhaustivas con **ICMP** y **TCP**, y todo funciona perfectamente. Lo que ya no he probado tanto ha sido el **UDP**, así que os invito a que probéis vosotros mismos los resultados. Por lo que os voy a explicar ahora, es posible que con **UDP** las cosas no funcionen tan bien, así que quizá deberíamos cambiar esas 2 reglas tan radicales por estas (pongo solo las de **FORWARD**, para **INPUT** podríamos copiarlas tal cual):

iptables -A FORWARD -p tcp --fragment -j DROP
iptables -A FORWARD -p icmp --fragment -j DROP
iptables -A FORWARD -p icmp --icmp-type destination-unreachable -j ACCEPT

¿Y esa última línea? ¿Se le ha pirado la pinza al chaval con tanta iptable? Pues no, tranquilos, que aún no me he vuelto loco.

Esa última línea es necesaria para que funcione la técnica que os voy a explicar a continuación que permite prescindir de la fragmentación IP.

PMTUD

Posiblemente este título os suene ya de algo, ya que en el artículo sobre **ICMP** expliqué a

grandes rasgos en qué consiste el **PMTUD** (**P**ath **MTU** **D**iscovery).

El **PMTUD** es un mecanismo relativamente simple que permite encontrar el **PMTU** de una ruta entre dos máquinas.

¿Y qué es el **PMTU**? Pues bueno, hasta el momento sabemos lo que es la **MTU**. La **MTU** (**M**aximum **T**ransmission **U**nit) es el tamaño máximo de paquete que puede transportar una red de una tecnología concreta (por ejemplo, *1500 bytes* para el caso de redes *Ethernet*).

El problema es que **Internet** es una red terriblemente **heterogénea**, donde conviven todo tipo de tecnologías de red diferentes. En el camino que puede haber entre dos máquinas conectadas a Internet puede haber routers que interconecten tecnologías de lo más variopintas.

Así, en el camino entre nuestro PC y un servidor web nos podemos encontrar fácilmente con que nuestros paquetes circulan por redes *Ethernet, PPPoE* (tecnología utilizada en *ADSL*), y muchas otras tecnologías diferentes. Cada una de las redes por las que pase el paquete tendrá su propia **MTU**.

Pero lo que a nosotros nos importa realmente a la hora de enviar nuestro paquete, si no lo queremos fragmentar, es cuál es **LA MENOR** de todas esas **MTU**s que nos encontraremos por el camino (**path**, en inglés). **A la menor de las MTUs que hay en un camino (path) entre dos máquinas, la denominamos Path MTU, o PMTU**. Si nuestros paquetes no superan en tamaño a la **PMTU**, no habrá ningún motivo por el que nuestros datagramas tengan que ser fragmentados para llegar a su destino.

Entonces, ¡aquí tenemos la solución para evitar la fragmentación! Nos basta con encontrar de alguna manera cuál es la **PMTU** de la ruta que hay entre nuestro PC y la otra máquina con la que queremos conectarnos. Una vez que la conocemos, serán los protocolos de nivel superior los que se encarguen de dividir sus datos de forma conveniente, en lugar de dejar esa responsabilidad a la capa IP.

Pero, ¿es que los protocolos que están por encima de IP también dividen sus datos?

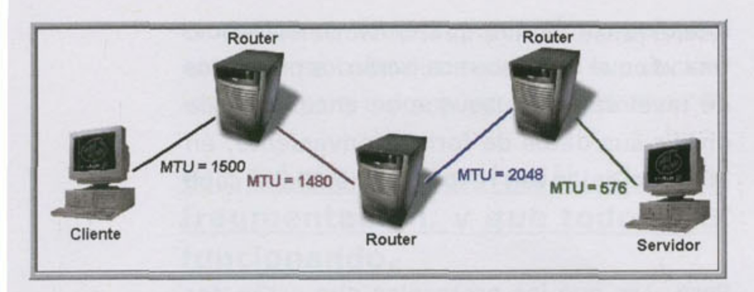
¡Pues claro! ¿Acaso pensáis que cuando queremos enviar un archivo de *650MBs* (como una *ISO* de un CD) lo que hace nuestra capa **TCP** es enviar de golpe los 650MBs a la capa IP para que ésta se apañe y los fragmente como pueda?

Esta claro que la capa TCP también tiene sus limitaciones y no puede andar trasteando con paquetes de cualquier tamaño. De hecho, algo sabemos acerca de esta "limitación", pues ya hablamos en este curso acerca de la **opción MSS** (**M**aximum **S**egment **S**ize) de la **cabecera TCP**. Precisamente en este campo **MSS** es donde se apoya el mecanismo de **PMTUD**.

Como ya vimos, la opción **MSS** se incluye sólo en **el primer paquete de una conexión TCP**, es decir, en el paquete que lleva el flag **SYN**. En el caso del **cliente**, será el paquete inicial de la conexión, que llevará solo el flag **SYN**, y en el caso del **servidor** será el segundo paquete de la conexión (primero que envía el servidor), que es el que lleva los flags **SYN y ACK**.

Este primer paquete es la primera pista para el mecanismo de **PMTUD.** Pero mejor vamos a ir viendo esto paso por paso con un ejemplo concreto.

Tenemos 2 máquinas: **cliente, y servidor**. **Cliente** establece una conexión **TCP** con **Servidor**, para lo cual tiene que enviar su **paquete TCP inicial** (con el flag **SYN**, y la opción **MSS**). Este paquete circulará a través de toda la ruta que une ambas máquinas.



Y la clave de todo está en cierto campo de este primer paquete, y más concretamente en un único bit. **El flag DF de todos los paquetes (incluyendo, por supuesto, este primero) tiene que estar activado.** Esta es la clave del funcionamiento del **PMTUD.**

Por tanto, con este primer paquete, además de intentar establecer la conexión (flag **SYN),** estamos haciendo dos cosas más.

Por una parte, estamos informando al **Servidor** del **tamaño de paquetes** que podemos manejar (opción **MSS**), y por otra estamos tanteando la **ruta** que hay entre nosotros y el servidor, ya que al activar el flag **DF**, nuestro paquete sólo podrá llegar a su destino si tiene un tamaño adecuado, es decir, si es menor o igual que la **PMTU** entre el **Cliente** y el **Servidor**.

Con respecto a la primera cuestión, la de informar al **Servidor** acerca de nuestra **MSS,** seremos un poco listos a la hora de escoger el valor de **MSS** que le daremos al **Servidor**. Imaginemos que nuestra capa **TCP** puede manejar buffers de hasta 65535 bytes.

En principio, éste podría ser el valor que daríamos al campo MSS, indicando que TCP puede procesar paquetes de hasta 64KB. Pero, ¿qué utilidad tendría esto si sabemos de antemano que otras capas inferiores no van a poder con ese paquete? Vale, nosotros no sabemos cual es la **PMTU**, es decir, no sabemos cómo de pequeños tendrán que ser los paquetes para poder recorrer toda la ruta, pero lo que sí que sabemos es cuál es **NUESTRA MTU**, es decir, el **tamaño máximo de paquete para la red a la que nosotros estamos conectados.**

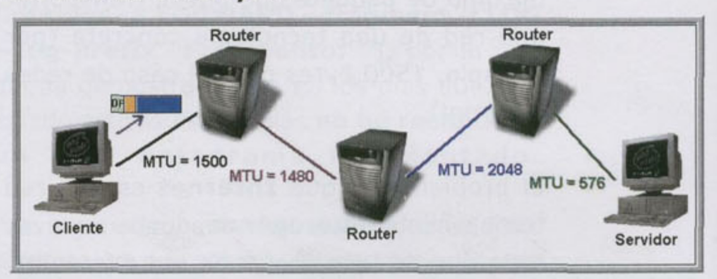
Por tanto, lo más lógico es partir ya en principio de una primera aproximación a la **PMTU**, ya que sabemos de antemano que la **PMTU** nunca podrá ser mayor que nuestra propia **MTU** (ya que la PMTU es la menor de todas las MTUs que haya en toda la ruta, incluida la nuestra, por supuesto).

Pongámonos en el caso de que nuestra red sea una **Ethernet**, que tiene una **MTU** de **1500 bytes**. ¿Debemos poner entonces **1500** en el campo **MSS**? ¡Pues **no**! ¡No tan rápido! Hay que tener en cuenta que **1500 bytes** es el tamaño máximo para un **datagrama,** es decir, para un **paquete IP**.
Pero la opción **MSS** no se refiere a paquetes IP, si no a **paquetes TCP**. Un paquete TCP es un paquete IP al cual le hemos quitado (o más bien, aún no le hemos añadido) la **cabecera IP**. Y el campo **MSS** *se refiere al tamaño de los* **datos** de un paquete TCP (es decir, sin la **cabecera TCP**).

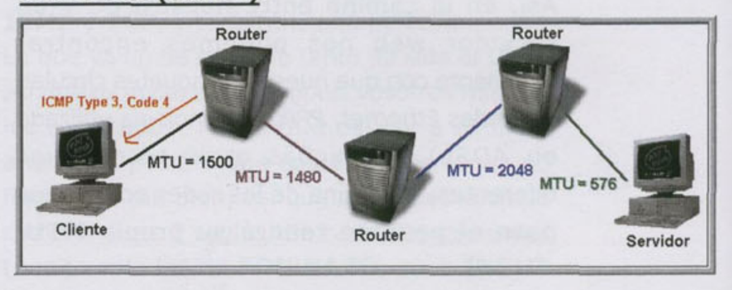
Por tanto, teniendo en cuenta que el tamaño de una **cabecera TCP** (sin opciones, claro) es de **40 bytes**, para calcular el valor de **MSS** tendremos que **restar 40 bytes a la MTU**. Es decir, en nuestro caso: **1500 -40 = 1460.**
Por tanto, en nuestro campo **MSS** tendremos que poner **1460** si queremos que nuestros paquetes TCP se ajusten a nuestra **MTU** una vez que sean convertidos en datagramas.

Vamos a ver qué ocurre ahora cuando nuestro primer paquete, con **MSS = 1460** y el flag **DF** activado, intenta llegar hasta el servidor.

En primer lugar, el paquete llega a la conexión que hay entre nuestro router y nuestro ISP. Esta conexión, como vemos, tiene una **MTU** de **1480 bytes**.



Como nuestro paquete mide 1500 bytes (1460 de TCP + 40 de la cabecera IP) y tenemos activado el flag DF, el paquete no podrá circular más allá, por lo que nuestro router nos lo rechazará, indicándonos el motivo con un mensaje **ICMP de tipo Destination Unreachable (type 3) y con código Fragmentation needed and DF set (code 4)**.



He aquí la necesidad de estas líneas que vimos antes:

iptables -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type destination-unreachable -j ACCEPT

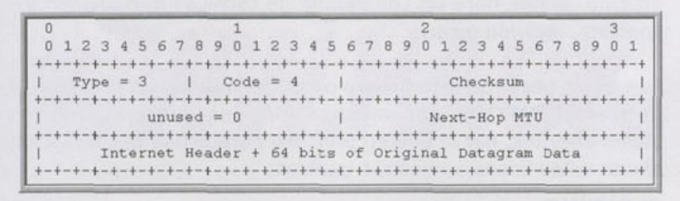
Ya que ante este mensaje ICMP, nuestra máquina tendrá que responder, sabiendo que de ahora en adelante los paquetes tendrán que ser más pequeños para poder pasar al menos el primer tramo del camino. Pero... **¿cómo de pequeños?**

Pues aquí es donde entra en juego un nuevo RFC, el **RFC 1191** (http://www.ietf.org/rfc/rfc1191.txt) que, como vemos, tiene por título precisamente **"Path MTU Discovery".**

Por lo que vimos en el artículo sobre **ICMP**, éste es el formato de un ensaje **ICMP Destination Unreachable:**

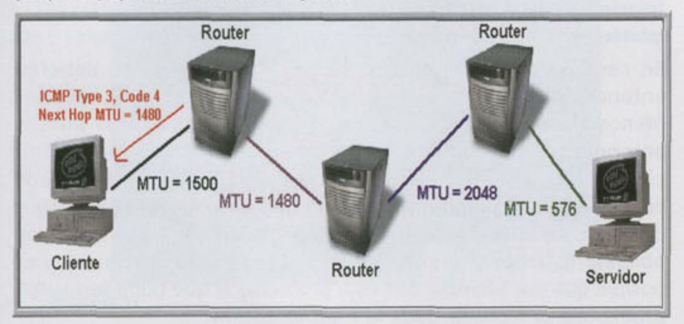
```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             unused                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Lo que especifica este RFC es un pequeño añadido a este formato, que consiste en utilizar parte del campo **UNUSED** para incluir el dato que nosotros necesitamos: la **MTU** de la red a la que hemos intentado acceder.



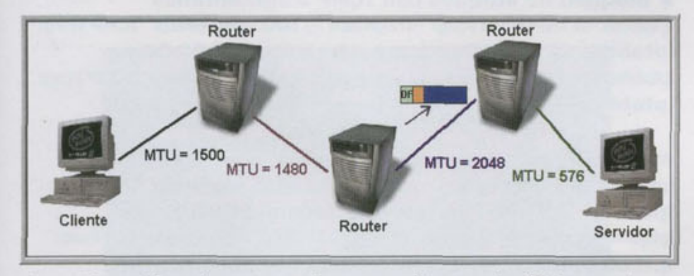
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 3    |   Code = 4    |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           unused = 0          |         Next-Hop MTU          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Datagram Data      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Por tanto, con el mensaje ICMP de nuestro router, nos llegará también el valor al que tenemos que ajustar nuestro nuevo paquete, que serán **1480 bytes**:

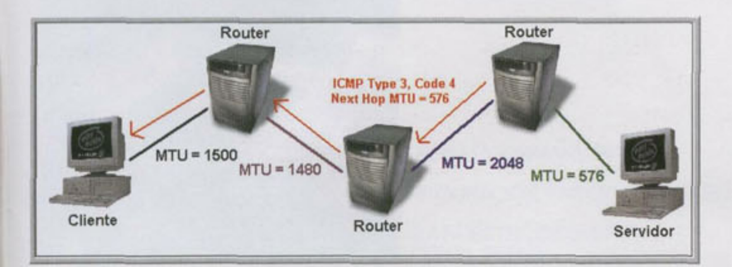


Como este paquete no ha llegado a su destino, tendremos que **reintentarlo**, pero esta vez, por supuesto, ajustando el tamaño de nuestro datagrama a **1480 bytes**. Por tanto, ahora nuestro **MSS** será de **1440 bytes.**

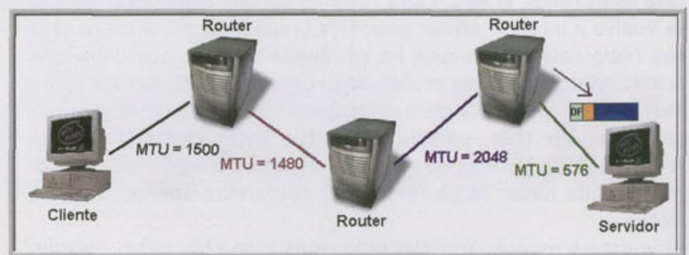
En el siguiente paso, nos encontramos con una red que tiene una **MTU** de **2048 bytes**. Nuestro paquete cabe de sobra por esa red, por lo que podrá atravesarla sin mayor problema:



A continuación, el paquete tiene que llegar a la última red que tenemos en nuestra ruta, la cual tiene una **MTU** de **576 bytes**. Por aquí no podrá pasar, así que el router de turno nos responderá con un **ICMP type 3 code 4**:



El mensaje ICMP llegará hasta nosotros, donde reajustaremos de nuevo el tamaño de nuestros datagramas, esta vez a **576 bytes**, y reenviaremos una vez más el paquete.



Esta vez si que ha llegado! El **Servidor** ve el campo **MSS** de nuestro paquete, en el cual se encuentra ya a estas alturas el resultado de nuestra búsqueda de la **PMTU** (**PMTUD**), que en este caso es de **576 bytes**. Con esto él ya sabe que en la ruta que hay entre él y nosotros hay que utilizar datagramas que se ajusten en tamaño a la PMTU. Como el valor de nuestra **MSS** es **536**, el servidor sabrá que sumando **40** a este número tiene ya la **PMTU** = 536 + 40 = **576 bytes**.

Por tanto, cuando nos responda con el paquete **SYN,ACK**, lo hará partiendo ya de un valor de **MSS = 536**. Por supuesto, su flag **DF** estará activado, como lo tiene que estar para **TODOS** los paquetes que circulen en ambos sentidos.

Por tanto, el mecanismo de PMTUD no se utiliza sólo en el establecimiento de la conexión, si no que el tamaño de los datagramas se puede ir ajustando dinámicamente para cada paquete.

Hay muchos motivos por los que la PMTU puede **cambiar** a lo largo del tiempo. Uno de los motivos principales es que se establezca una nueva conexión que, aunque sea entre las dos mismas máquinas, utilice un tipo de servicio (**TOS**) diferente. Muchas veces los routers modifican la ruta entre dos máquinas en función del TOS, por lo que, al haber una nueva ruta, el PMTU puede cambiar.

Por tanto, el sistema operativo de cada máquina tiene que guardar un **registro** *de las PMTUs encontradas para las rutas ya conocidas*. Una **ruta** incluirá no sólo **una dirección IP de origen y una de destino**, si no también **un tipo de servicio** (TOS).

Como todos los paquetes tendrán siempre el bit **DF** activado, en cuanto algún paquete sea rechazado con un mensaje **ICMP type 3 code 4**, ya sabremos que tenemos que reducir el tamaño de nuestros datagramas. Aunque no podamos ya cambiar el valor *del campo* **MSS** (*ya que esta opción es enviada* **SÓLO** en el **establecimiento** de la conexión **TCP**), sí que podremos ir ajustando dinámicamente el tamaño de los datagramas, igual que irá haciendo el otro extremo de la comunicación según se vaya encontrando con los mismos "obstáculos".

Igual que puede haber limitaciones que aparezcan dinámicamente en el tamaño de la PMTU, también puede ocurrir el caso contrario: que la PMTU **se incremente** en un momento dado. Si la ruta cambia hacia una ruta "mejor", es decir, con una PMTU más grande, estaremos desaprovechando la capacidad de nuestra ruta

al seguir insistiendo en enviar paquetes tan pequeños como los que debíamos enviar antes en función de la PMTU antigua.

Para estos casos, el **RFC 1191** recomienda que cada cierto tiempo se vuelva a intentar enviar paquetes grandes, a ver si cuela esta vez (sólo colará si la ruta ha cambiado sin que nos diéramos cuenta, claro). Como la probabilidad de que cambie la ruta no es muy alta, no merece la pena estar reintentando paquetes grandes cada dos por tres, ya que supondría estar constantemente reajustando el tamaño de nuestros datagramas. Por eso, el RFC recomienda hacer estos reintentos aproximadamente cada 10 minutos.

Si vuestras mentes son tan retorcidas como las mías, habréis encontrado aquí una idea para un ataque **DoS.**

No he visto prácticamente nada documentado acerca de este tipo de ataque. Únicamente he encontrado un documento que compendia los ataques que explotan el protocolo ICMP, y comentaban como "curiosidad" la posibilidad de este ataque, indicando que no ha sido probado.

En el próximo artículo os contaré mis resultados prácticos con este experimento.

Pero bueno, ideja de enrollarte ya, y cuéntanos en qué consiste esa técnica DoS que te has "inventado"!

Pues imaginad qué pasaría si se nos ocurriese **suplantar a uno de los routers** que hay en el camino entre dos máquinas. Hacemos un simple **ip-spoofing** para que la IP de origen de nuestro paquete sea la del router, y enviamos a una de las dos máquinas (la que queramos DoSear) un mensaje **ICMP type 3 code 4** trucado a mano, en el cual le haremos creer que la **PMTU** ha cambiado a un valor todo lo bajo que podamos indicarle.

En teoría podríamos poner aquí un **0**, por lo que se creería que no puede enviar ni un mísero byte, y no podría continuar enviando ningún dato. En cambio, el **RFC 1191** obliga a que el valor mínimo para este campo sea de **68 bytes**. Ahora bien, una cosa es lo que diga el RFC y otra lo que implementen los programadores de los sistemas, así que habría que probar si realmente alguien ha hecho caso a esta recomendación del mínimo de 68 bytes.

En cualquier caso, aunque se sigan las recomendaciones del RFC 1191, podríamos reducir la PMTU de una conexión hasta en 68 bytes.

Por poner un ejemplo real, la PMTU que me suelo encontrar yo prácticamente siempre es de **1500 bytes**, por lo que por cada 1460 bytes de datos, estamos enviando 1500 bytes (sumando los 40 bytes de cabeceras).

En cambio, si reducimos a **68 bytes** la PMTU, por cada 28 bytes de datos estaremos enviando 68 bytes (al sumar los 40 de las cabeceras), por lo que si bien antes un 97% de cada paquete eran datos, ahora sólo un 41% de cada paquete serán datos.

Conseguiríamos teóricamente reducir así prácticamente a la mitad el ancho de banda de una conexión enviando tan sólo un único paquete spoofeado.

Como el RFC 1191 recomienda que no se revisen los incrementos en la PMTU en menos de **5 minutos**, podremos mantener este ataque sin que nos suponga a nosotros ningún consumo de ancho de banda, enviando el mismo paquete, por ejemplo, una vez por minuto.

Los detalles de este ataque espero poder mostrarlos en el próximo artículo, donde haré un compendio de técnicas que utilicen ip-spoofing, incluida ésta.

Aún así, a pesar de este peligro potencial que se nos presenta al utilizar el **PMTUD,** son muchas más las ventajas que los inconvenientes, por lo que en principio parece una buena opción añadir estas líneas a nuestras iptables:

```
# PMTUD
iptables -A INPUT -p tcp --fragment -j DROP
iptables -A FORWARD -p tcp --fragment -j DROP
iptables -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type destination-unreachable -j ACCEPT
```

En caso de que veamos que algo no funciona como debería, entonces tendríamos que revisar nuestras iptables y hacerlas menos restrictivas, tal y como fuimos viendo en los puntos anteriores, donde comentábamos técnicas concretas para filtrar cada tipo de ataque específico. De encontrar algún problema lo encontraríamos en algún protocolo que utilizase grandes flujos de datos, y que por algún motivo requiriese fragmentación obligatoriamente. Yo no utilizo **NFS**, pero por lo que he leído es posible que por ejemplo con este protocolo sí que pudiesen surgir problemas al rechazar toda la fragmentación.

Resumiendo

Esta es la solución que propongo yo para implementar un firewall con iptables protegido contra los ataques de fragmentación.

Por supuesto, estoy abierto a todo tipo de mejoras, dudas, críticas, o comentarios de cualquier tipo:

```
# Bloqueo de ataques con ICMP fragmentados
iptables -A INPUT -p icmp --fragment -j LOG --log-prefix "ICMP Frag: "
iptables -A INPUT -p icmp --fragment -j DROP
iptables -A FORWARD -p icmp --fragment -j LOG --log-prefix "ICMP Frag: "
iptables -A FORWARD -p icmp --fragment -j DROP

# PMTUD
iptables -A INPUT -p tcp --fragment -j LOG --log-prefix "TCP Frag: "
iptables -A INPUT -p tcp --fragment -j DROP
iptables -A FORWARD -p tcp --fragment -j LOG --log-prefix "TCP Frag: "
iptables -A FORWARD -p tcp --fragment -j DROP
iptables -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type destination-unreachable -j ACCEPT
```

Autor: PyC (LCo)



CONSIGUE LOS NUMEROS ATRASADOS EN:

WWW.HACKXCRACK.COM



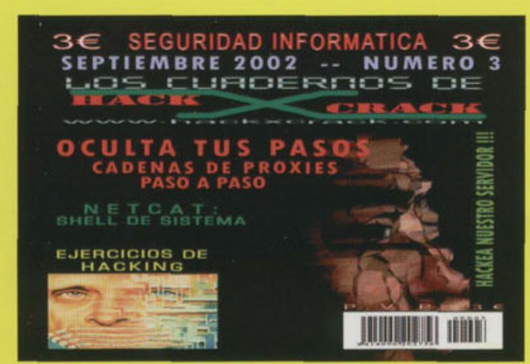
NÚMERO1:

-CREA TU PRIMER TROYANO INDETECTABLE POR LOS ANTIVIRUS.
-FLASHFXP: SIN LÍMITE DE VELOCIDAD.
-FTP SIN SECRETOS: PASVMODE.
-PORT MODE/PASV MODE Y LOS FIREWALL: LA UTILIDAD DE LO APRENDIDO.
-TCP-IP:INICIACIÓN (PARTE 1).
-EL MEJOR GRUPO DE SERVIDORES FTP DE HABLA HISPANA.
-EDONKEY 2000 Y SPANISHARE.
-LA FLECHA ÁCIDA.



NÚMERO 2:

-CODE/DECODE BUG: INTRODUCCIÓN.
-CODE/DECODE BUG: LOCALIZACIÓN DEL OBJETIVO.
-CODE/DECODE BUG: LÍNEA DE COMANDOS.
-CODE/DECODE BUG: SUBIENDO ARCHIVOS AL SERVIDOR REMOTO.
-OCULTACIÓN DE IP: PRIMEROS PASOS.
-LA FLECHA ÁCIDA: LA SS DIGITAL.
AZNAR AL FRENTE DE LA SS DEL SIGLO XXI.



NÚMERO 3:

-PROXY: OCULTANDO NUESTRA IP.
ASUMIENDO CONCEPTOS.
-PROXY: OCULTANDO NUESTRA IP.
ENCADENANDO PROXIES.
-PROXY: OCULTANDO NUESTRA IP.
OCULTANDO TODOS NUESTROS PROGRAMAS TRAS LAS CADENAS DE PROXIES.
-EL SERVIDOR DE HACKXCRACK: CONFIGURACIÓN Y MODO DE EMPLEO.
-SALA DE PRACTICAS: EXPLICACIÓN.
-PRÁCTICA 1ª: SUBIENDO UN ARCHIVO A NUESTRO SERVIDOR.
-PRÁCTICA 2ª: MONTANDO UN DUMP CON EL SERV-U.
-PRÁCTICA 3ª: *CODE/DECODE* BUG. LÍNEA DE COMANDOS.
-PREGUNTAS Y DUDAS.



NÚMERO 7:

- PROTOCOLOS: POP3
- PASA TUS PELÍCULAS A DIVX III (EL AUDIO)
- PASA TUS PELÍCULAS A DIVX IV (MULTIPLEXADO)
- CURSO DE VISUAL BASIC: LA CALCULADORA
- IPHXC: EL TERCER TROYANO DE HXC II
- APACHE: UN SERVIDOR WEB EN NUESTRO PC
- CCPROXY: IV TROYANO DE PC PASO A PASO
- TRASTEANDO CON EL HARDWARE DE UNA LAN